




Machine Learning for Physical Sensing & Perception

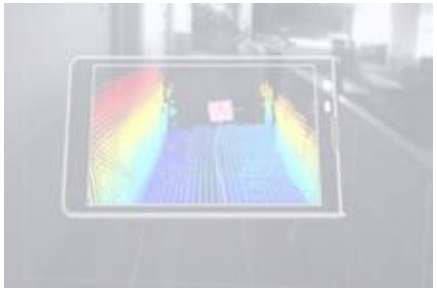
CMSC 818V : Fall 2026

Lecture 5:
From Sensor Arrays to Learned Spatial Inference

Nirupam Roy

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Tu-Th 12:30 – 1:45pm
IRB 2107



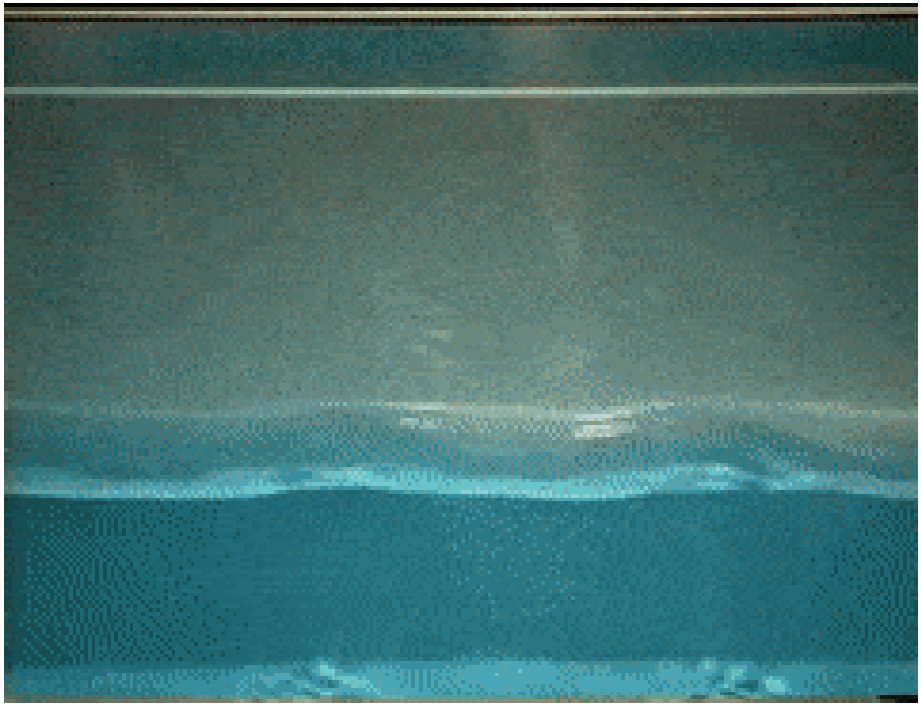

$$\boxed{\text{Physical system} \rightarrow \text{forward model} \rightarrow \text{inverse problem} \rightarrow \text{optimization} \rightarrow \text{learning}}$$

$$\boxed{\text{Physics} \rightarrow \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n} \rightarrow \text{Optimization} \rightarrow \text{Learning}}$$

Waves: The signals of the space-time



Waves: Transfer of energy, not mass



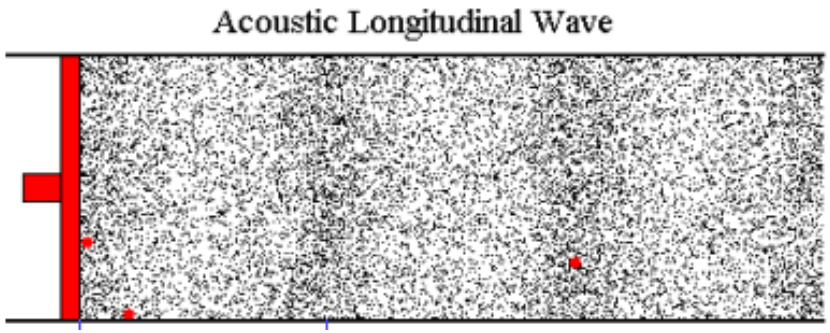
Mechanical Wave

Sound

Physical vibrations

Ripples in water

...



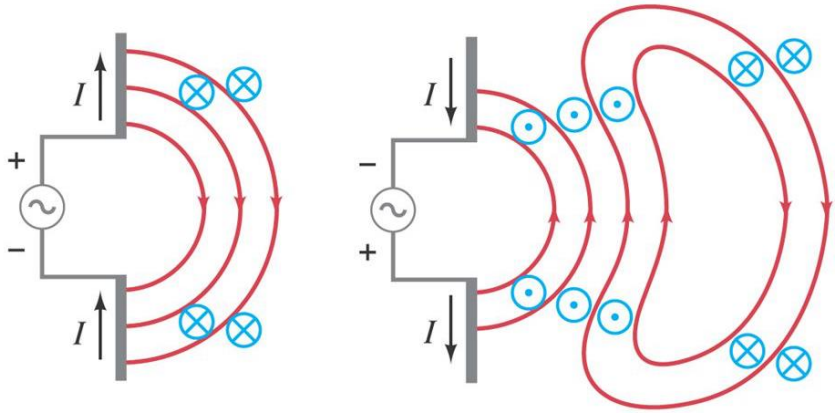
Electromagnetic Wave

Visible light

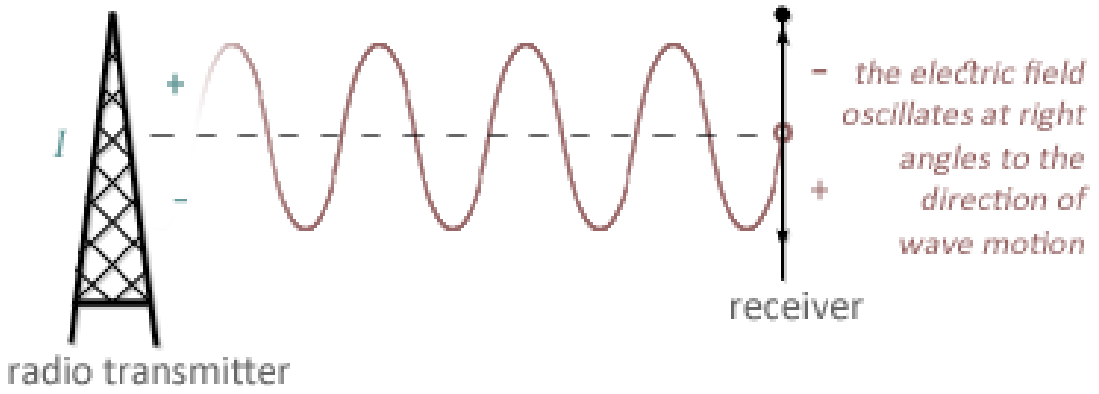
WiFi signal

Infrared

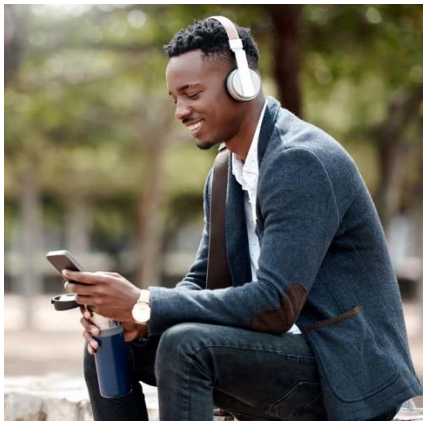
...



Waves made this possible







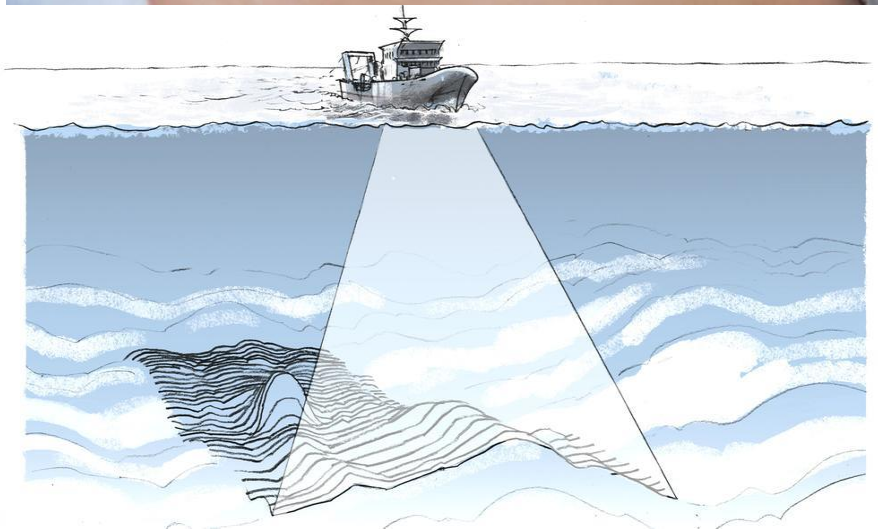


Waves made this possible

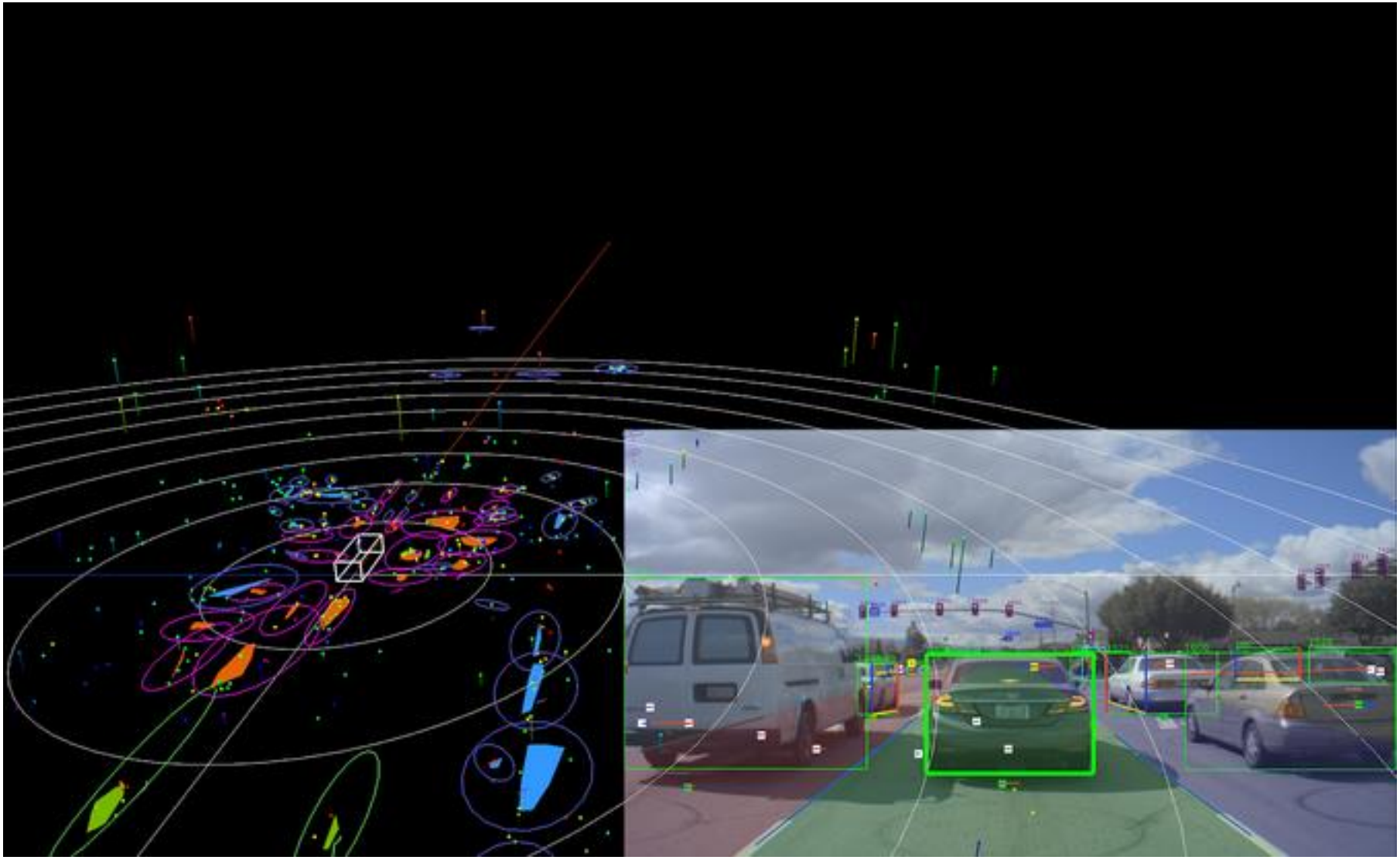








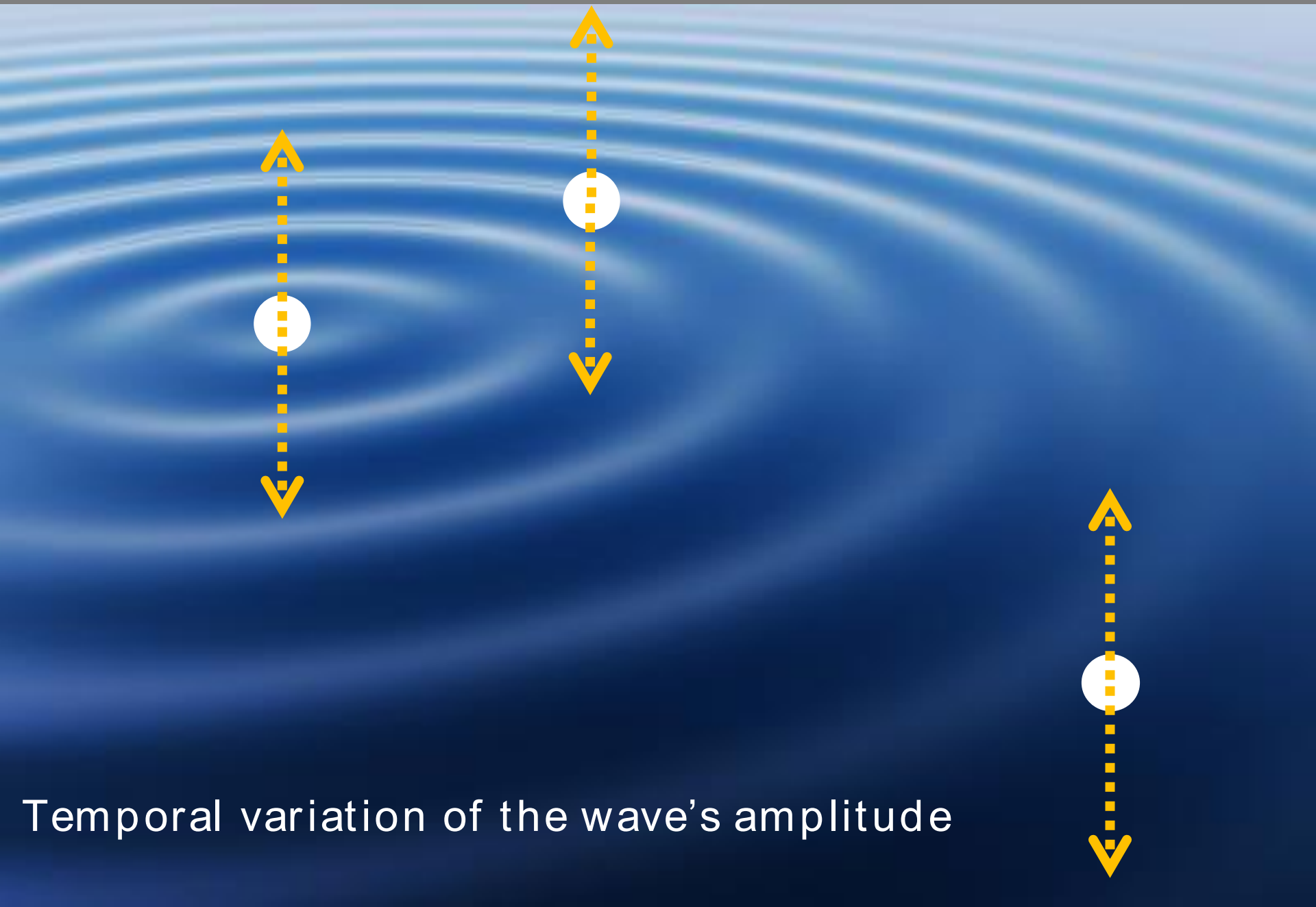
Waves made this possible



Waves in time and space



Waves in time and space

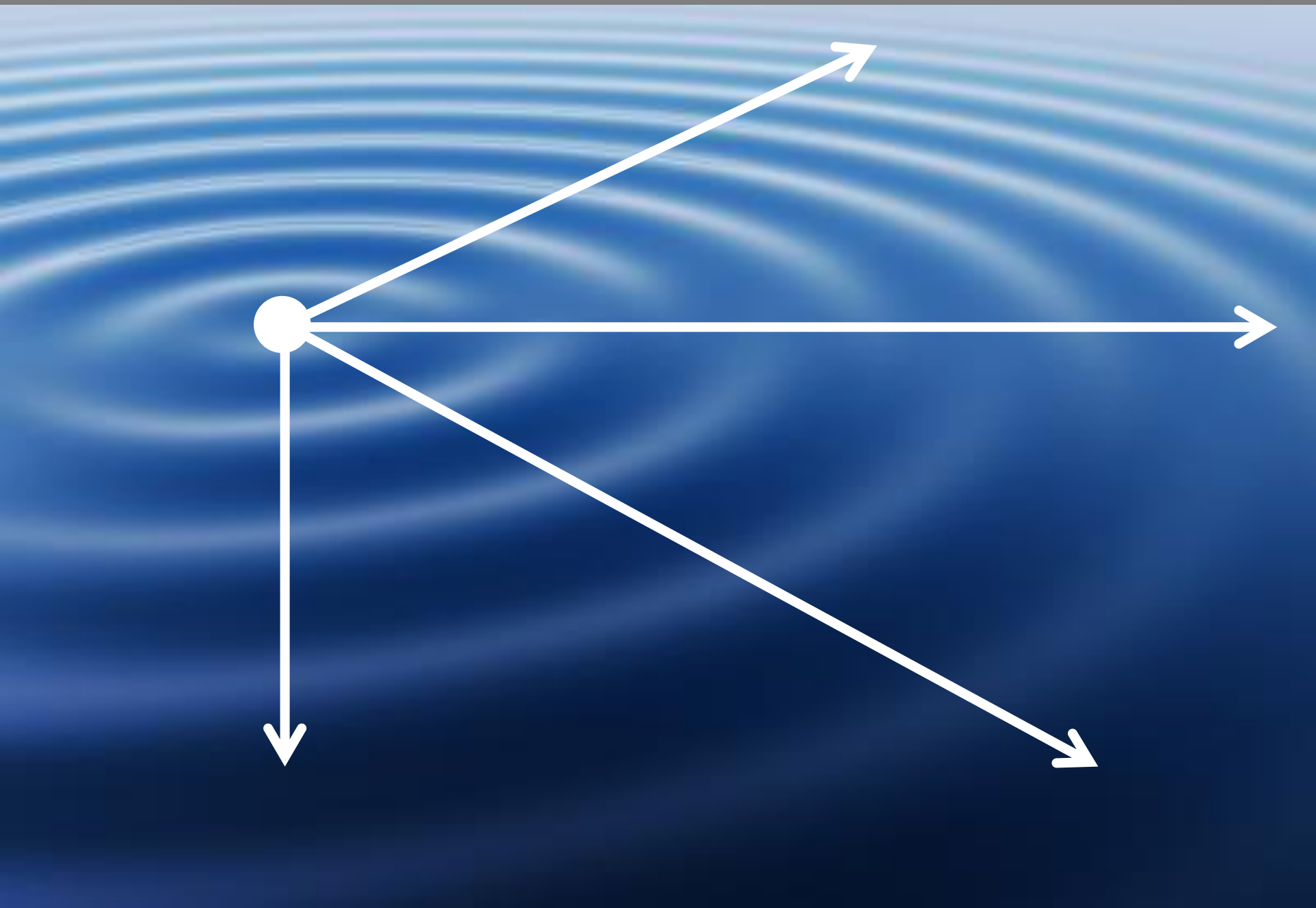


Temporal variation of the wave's amplitude

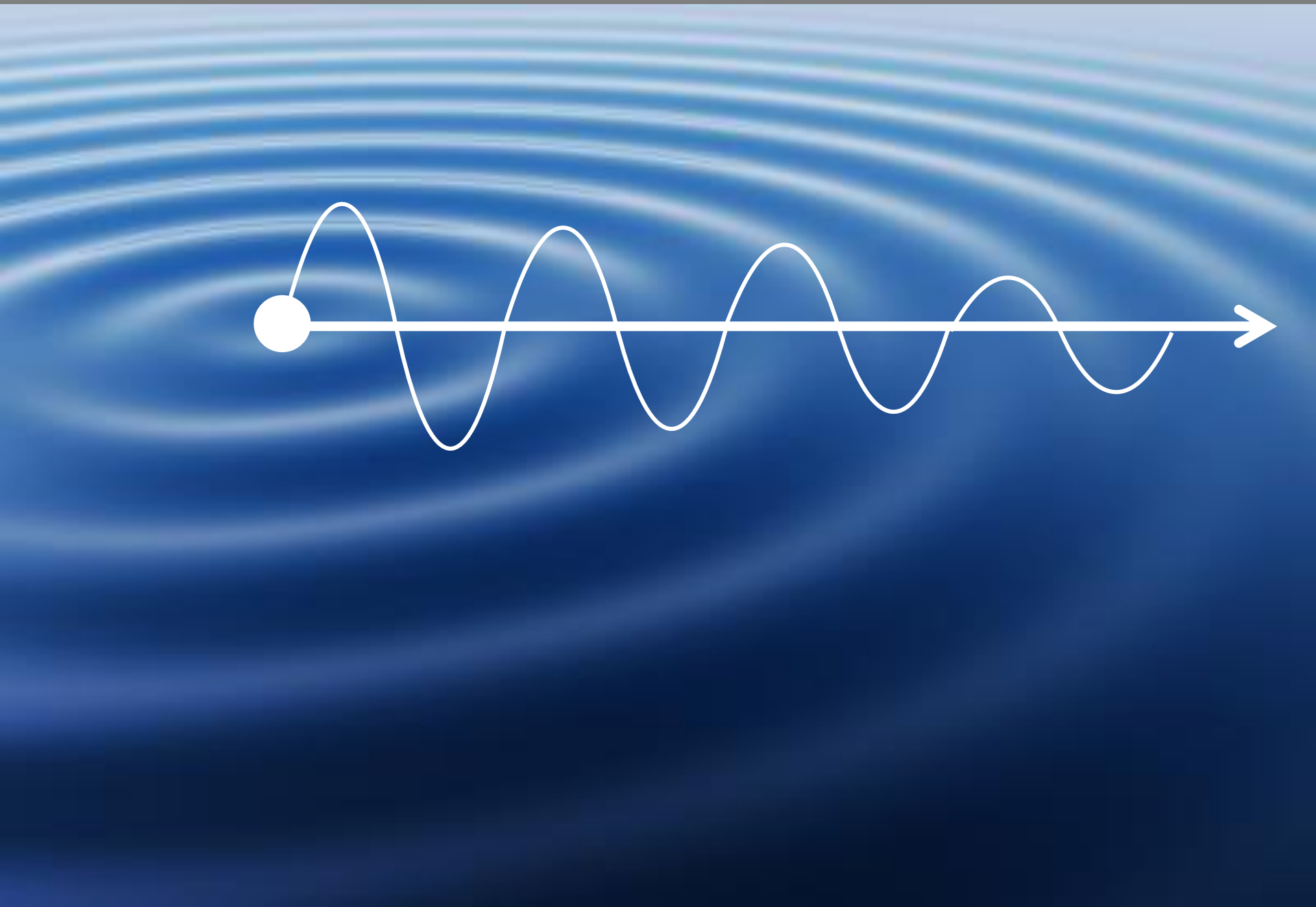
Waves in time and space



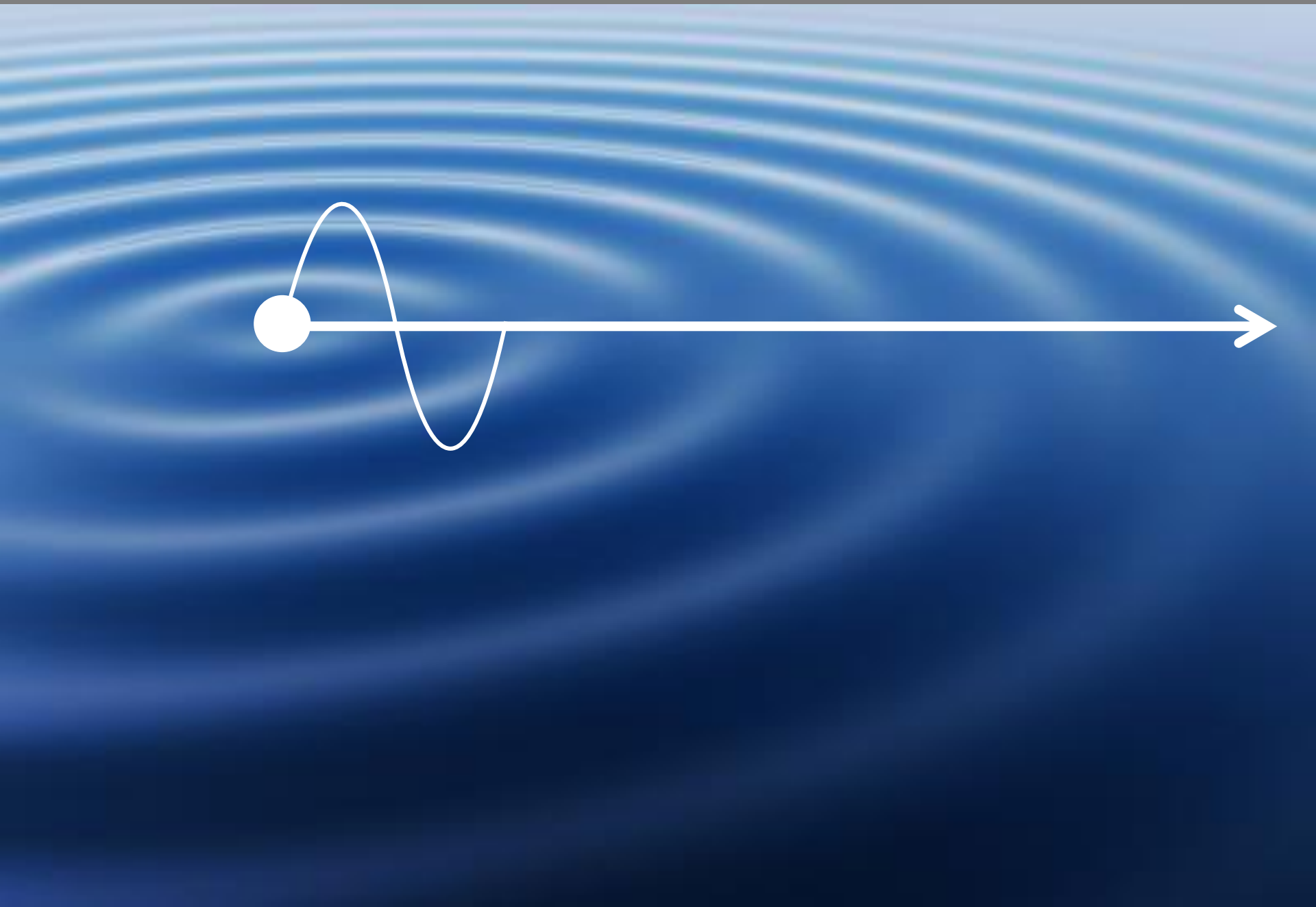
Waves in time and space



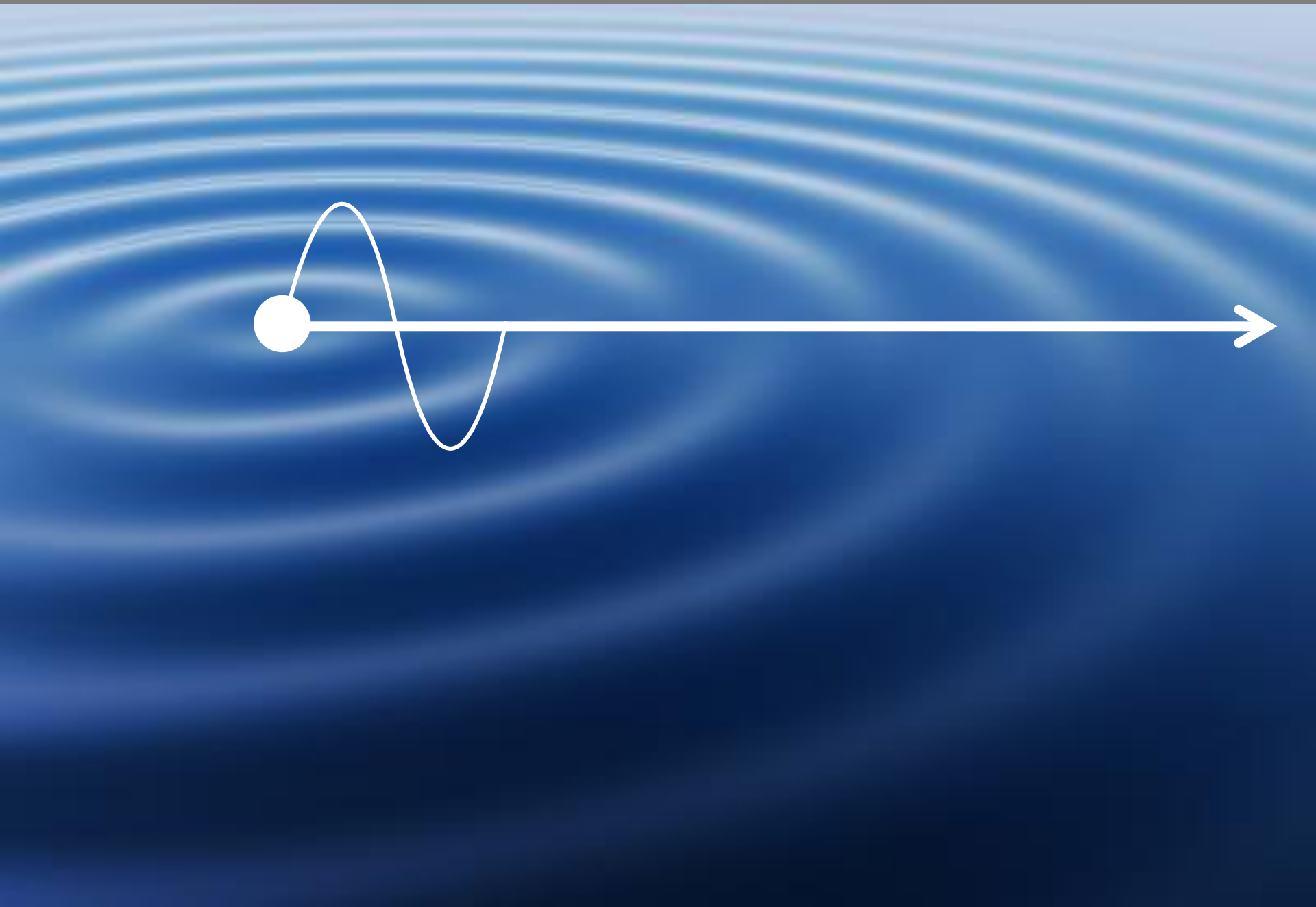
Waves in time and space



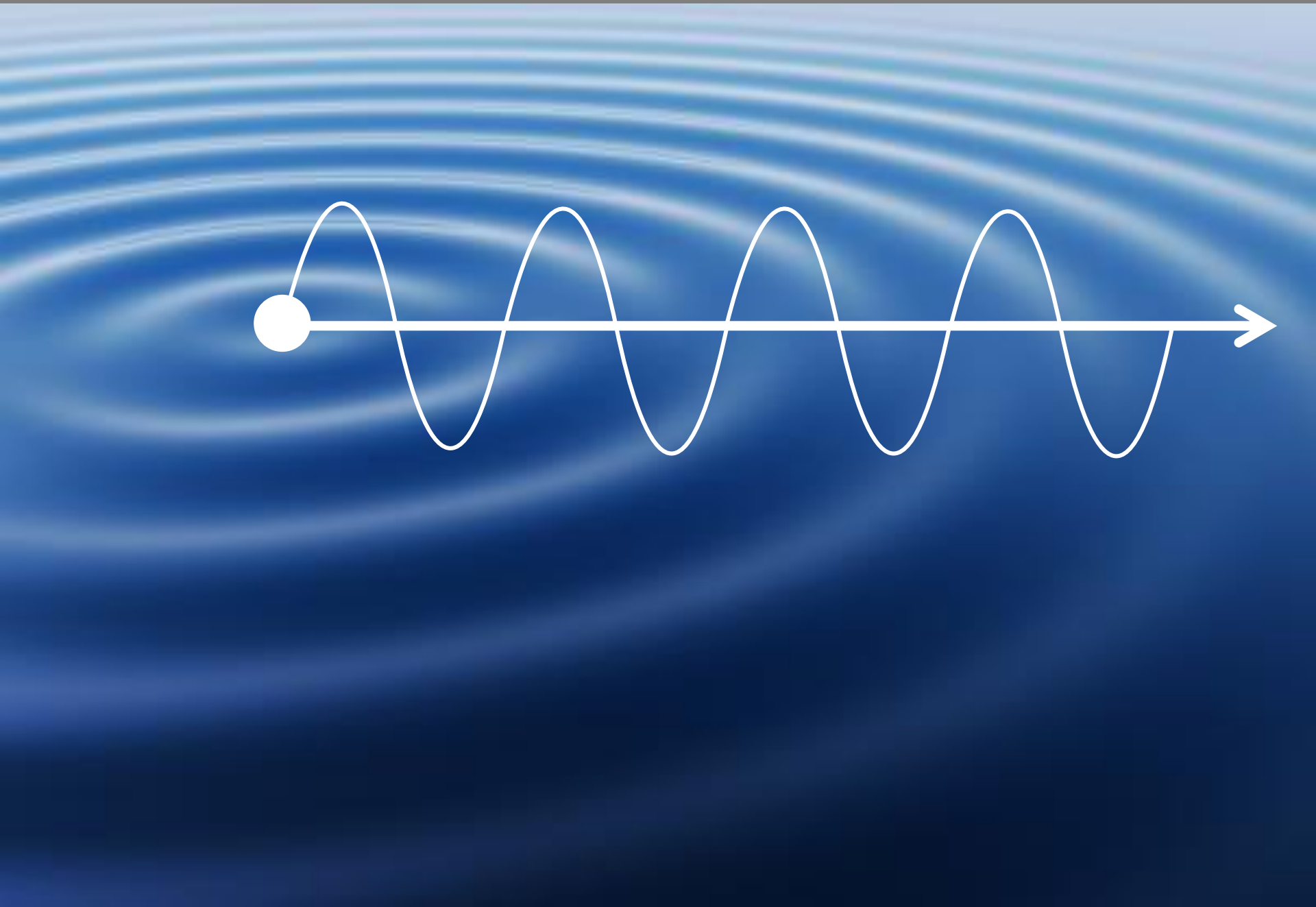
Waves in time and space



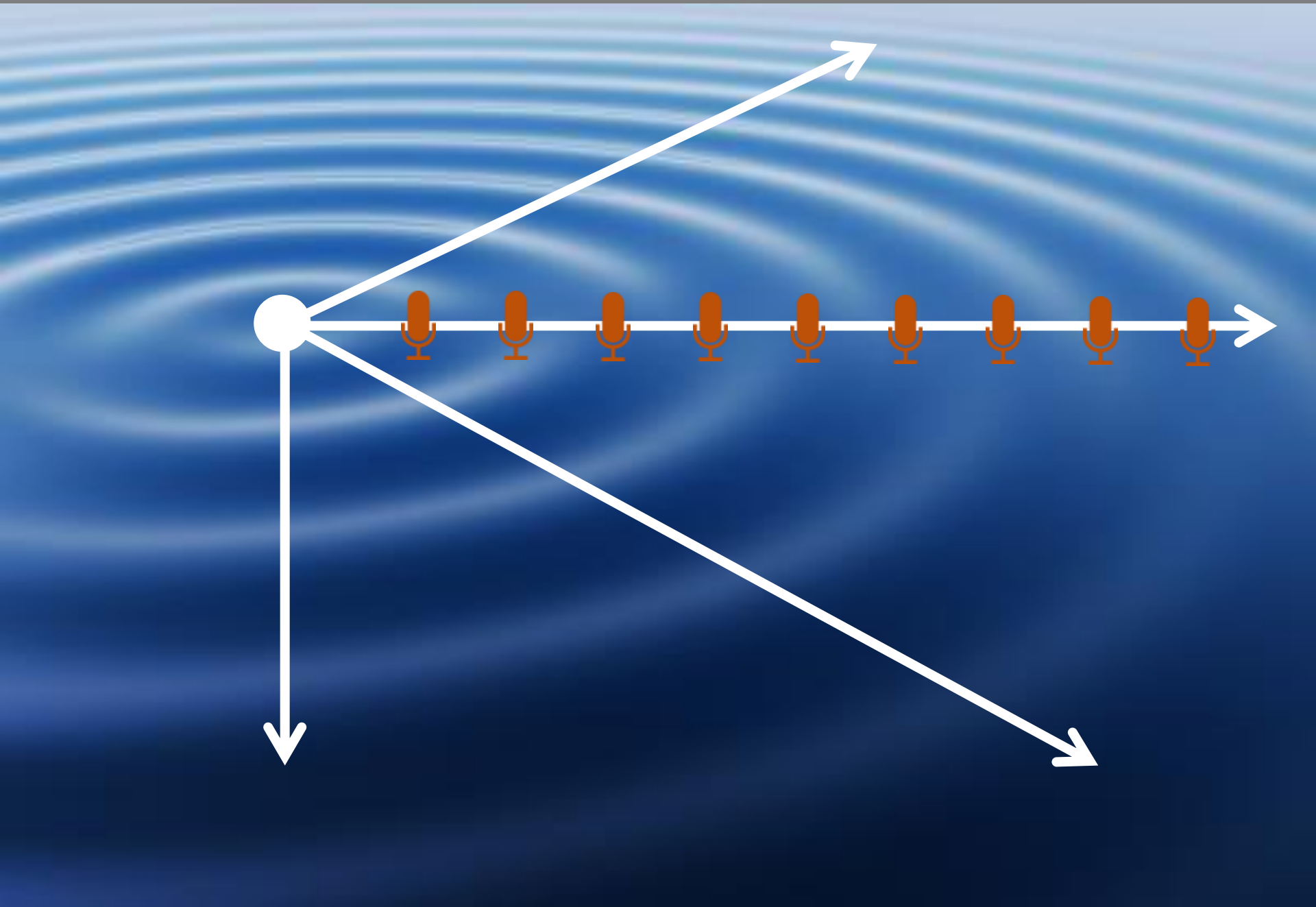
Waves in time and space



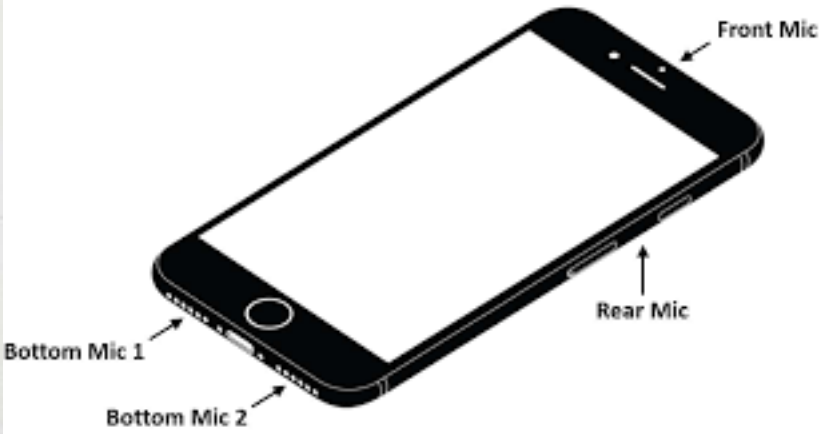
Equation of waves in time and space



Equation of waves in time and space



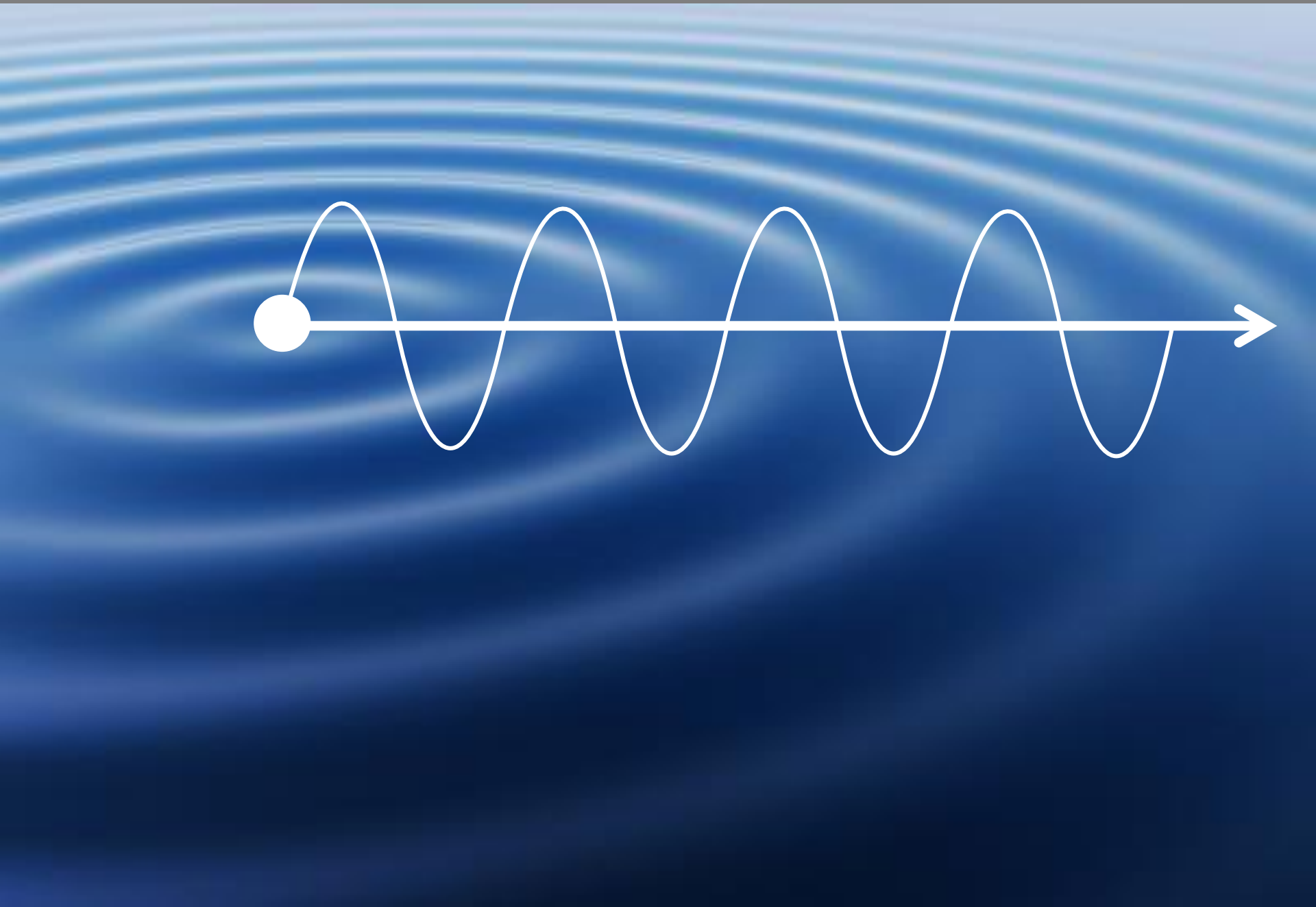

PCBA:31-000831
PCB: 30-000790 REV 12


Front Mic
Rear Mic
Bottom Mic 1
Bottom Mic 2

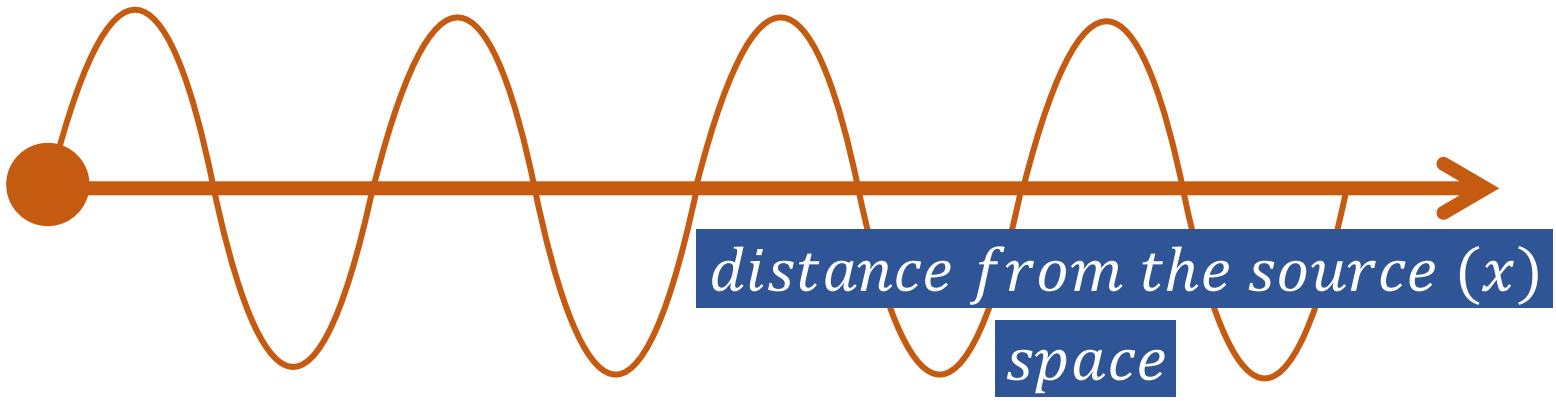




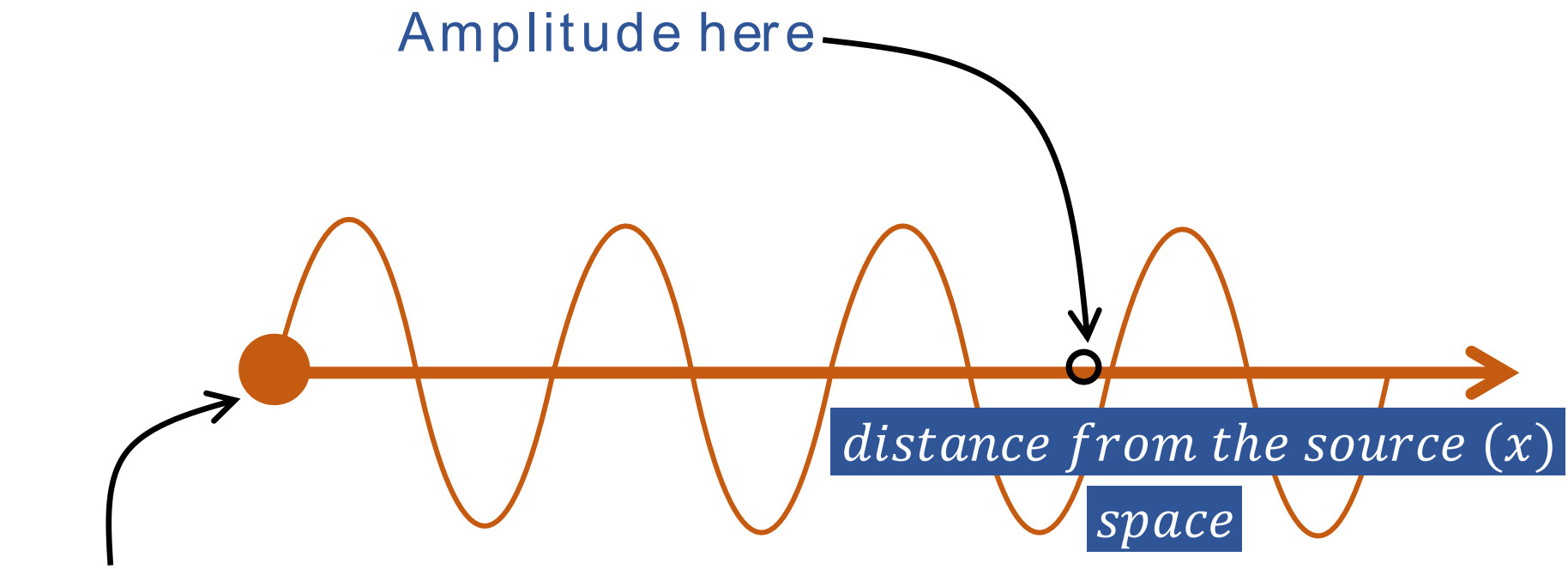
Equation of waves in time and space



Equation of waves in time and space

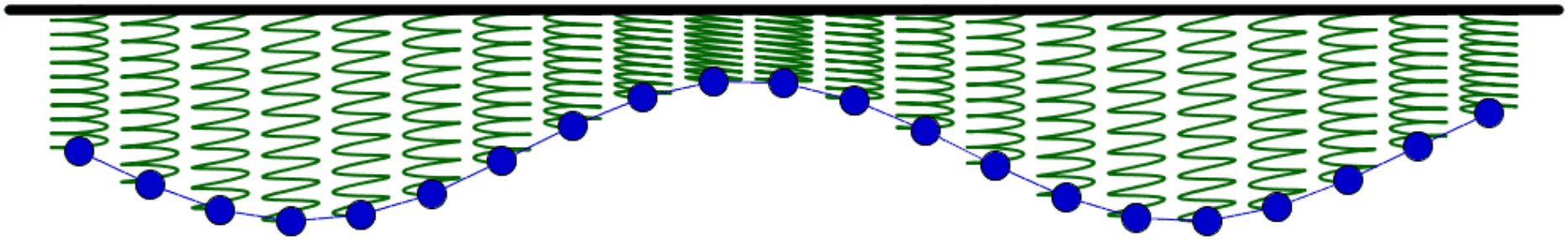


Equation of waves in time and space

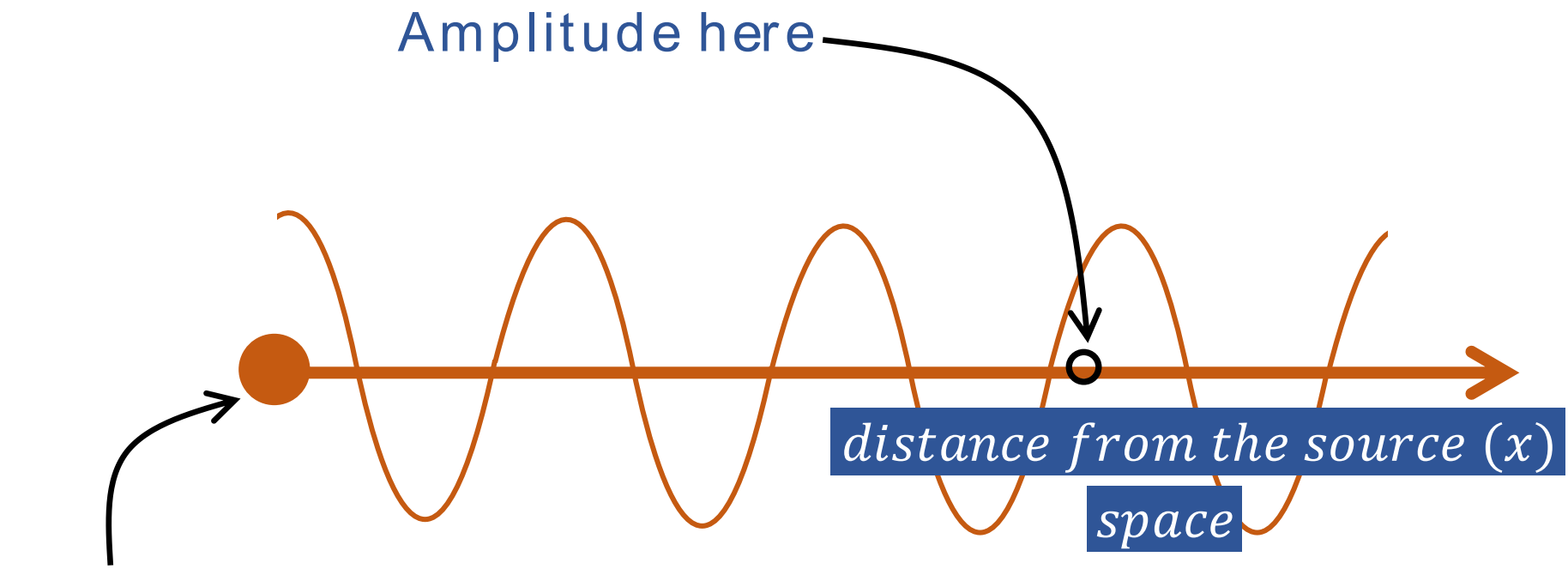




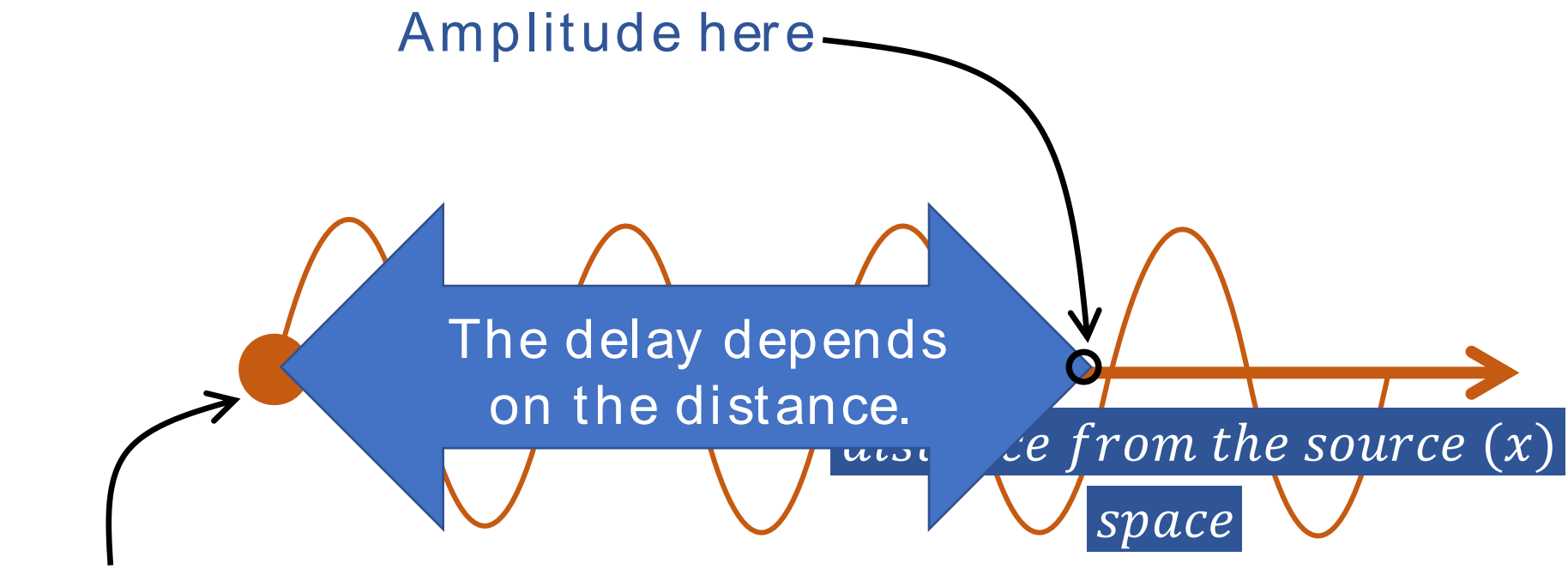




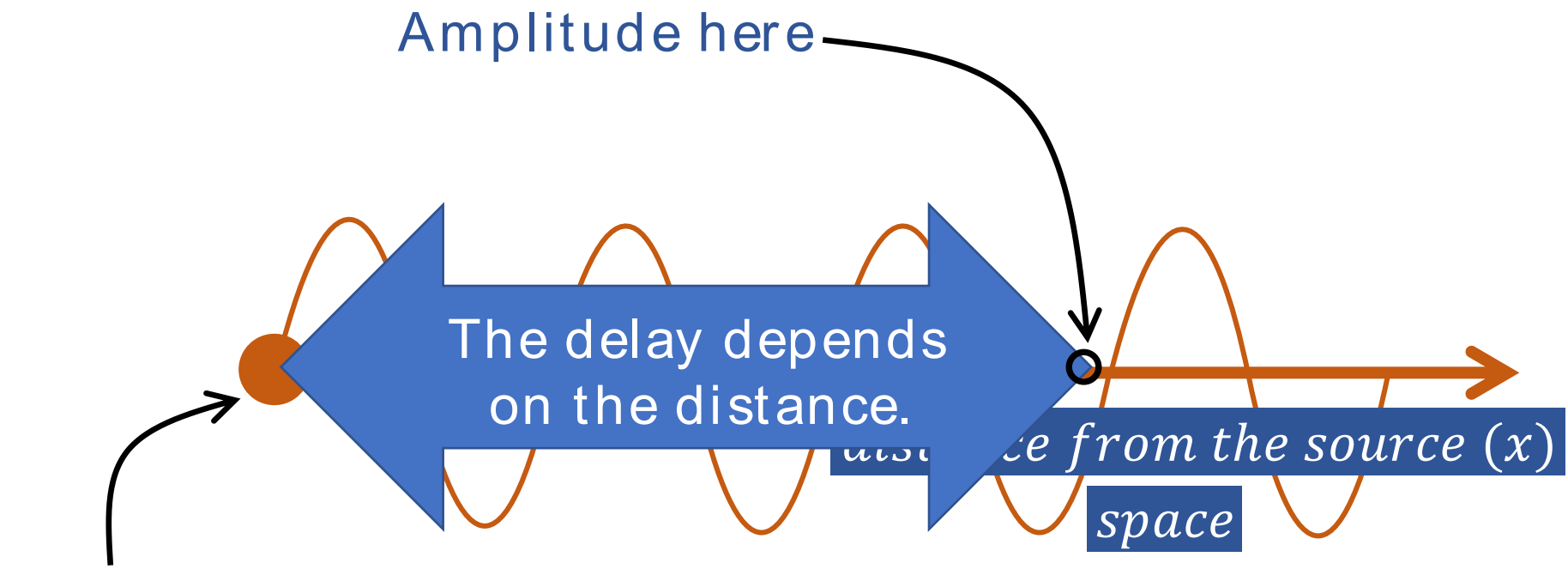
Equation of waves in time and space



Equation of waves in time and space

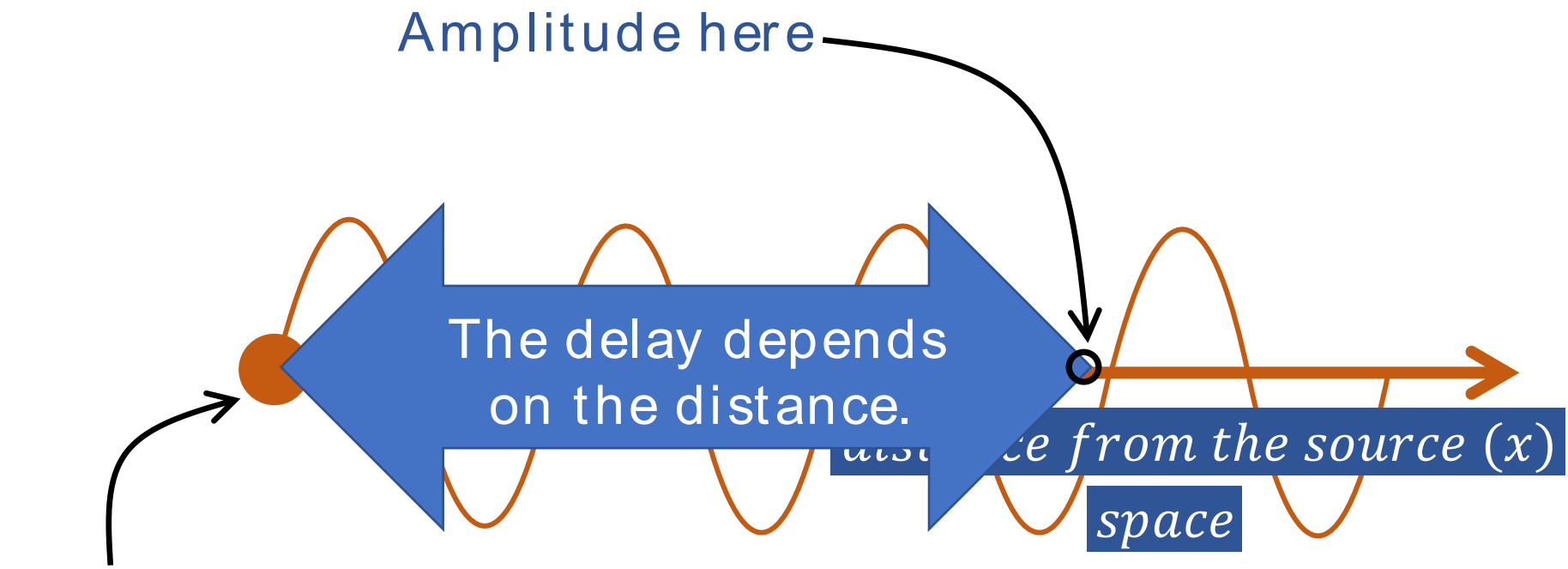


Equation of waves in time and space



$$\Psi(t) = A \,.\, \sin(2\pi f t)$$

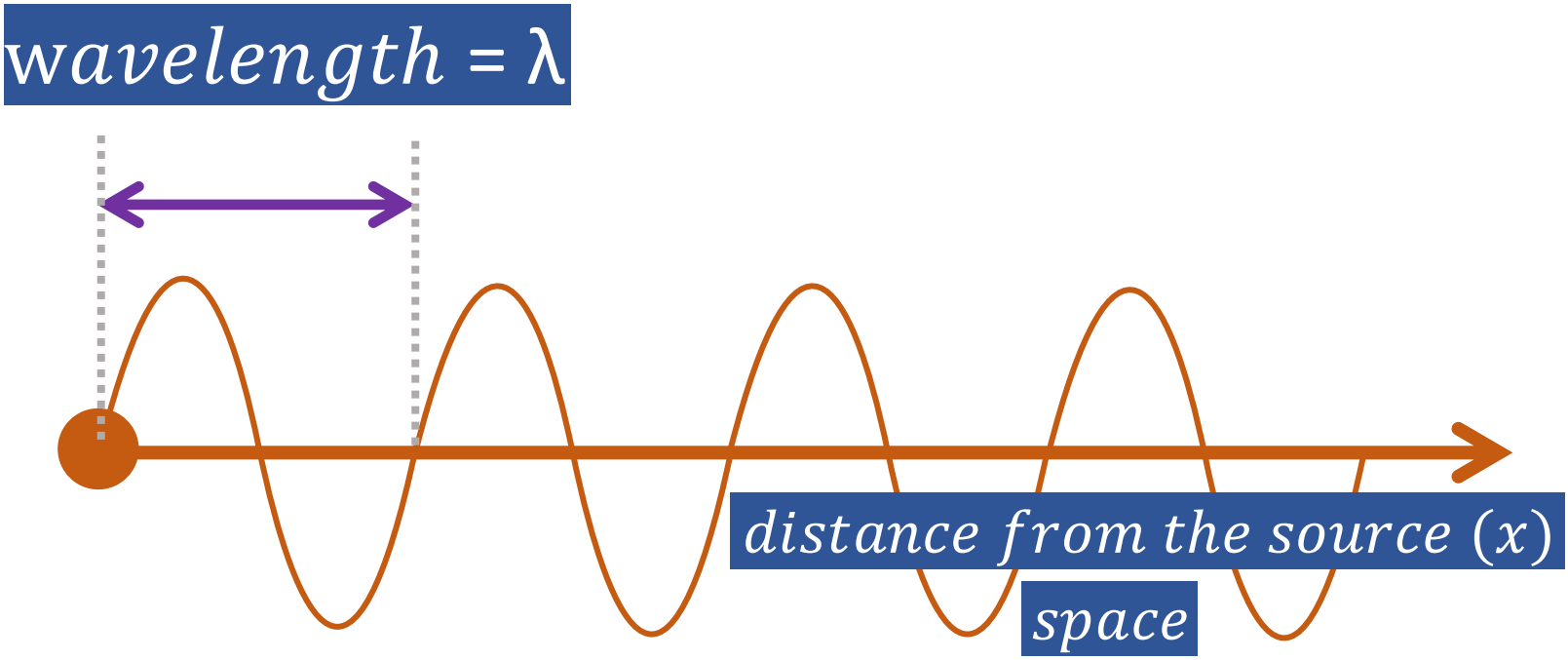
Equation of waves in time and space



$$\Psi(\mathrm{t},\,\mathrm{x}) = \mathrm{A}\,.\sin(2\pi f t - \theta(x))$$

Why is this negative?

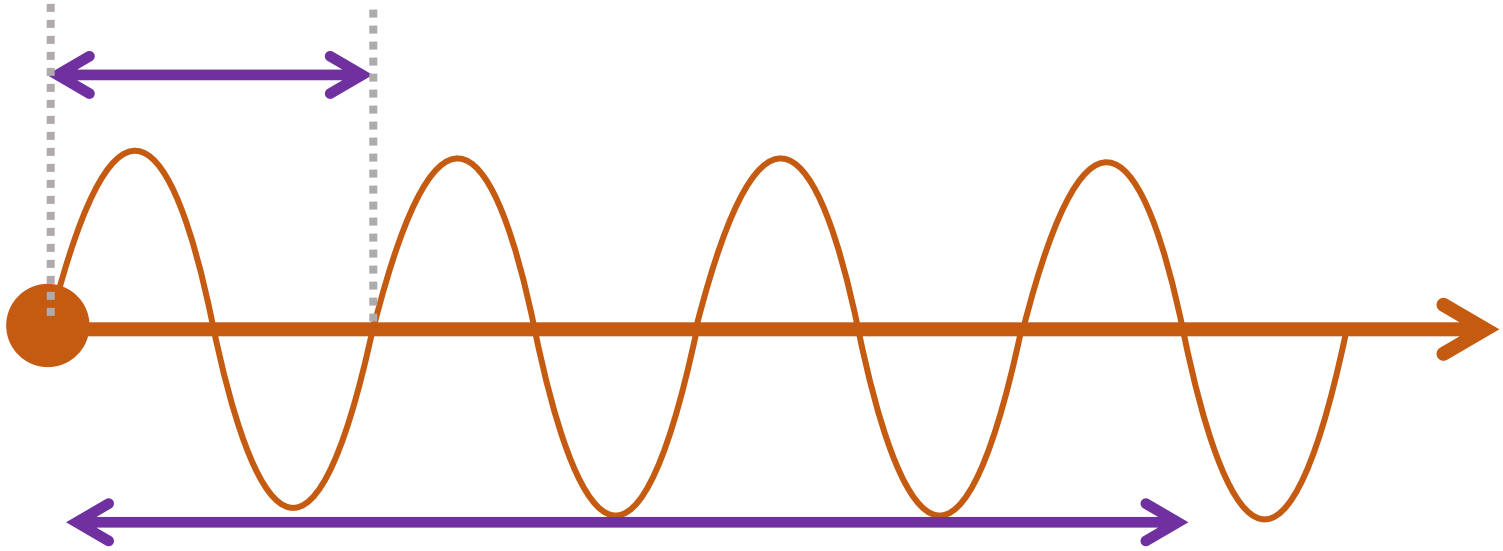
Equation of waves in time and space



$$\Psi(\mathrm{t},\ \mathrm{x}) = \mathrm{A}\,.\sin(2\pi f t - \theta(x))$$

Equation of waves in time and space

$wavelength = \lambda$



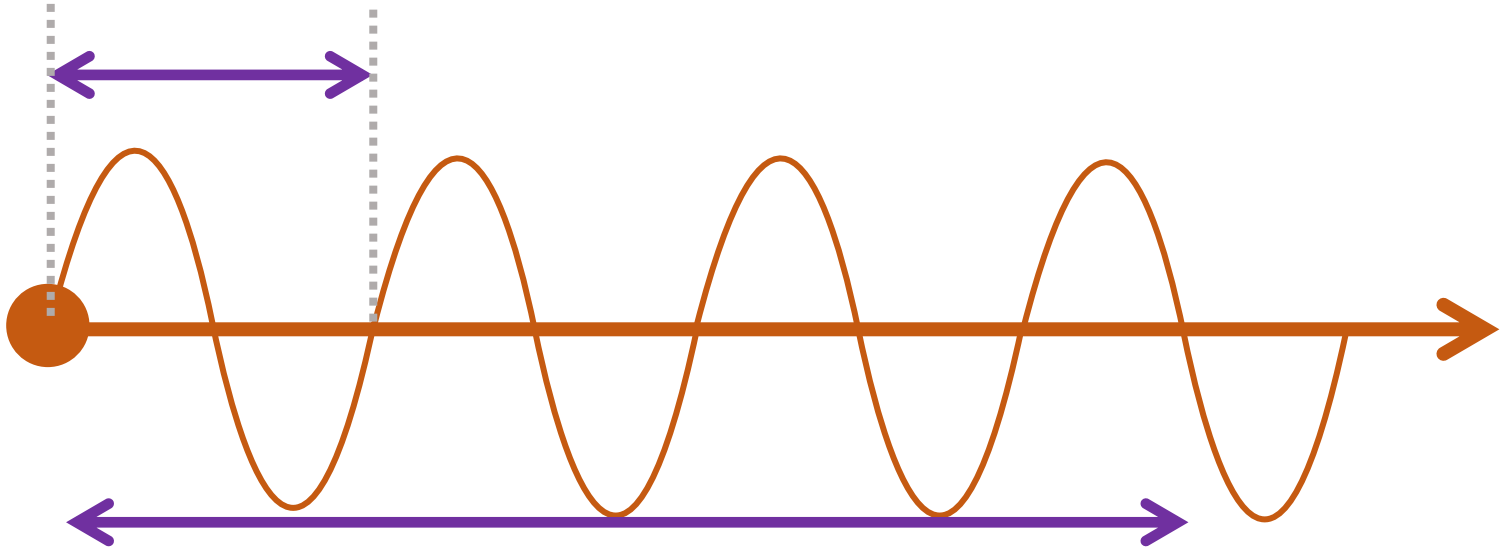
f *number of cycles per second*

C *meters of distance per second*
(speed of the wave)

$distance\ per\ cycle = \lambda = C / f$

Equation of waves in time and space

$wavelength = \lambda$



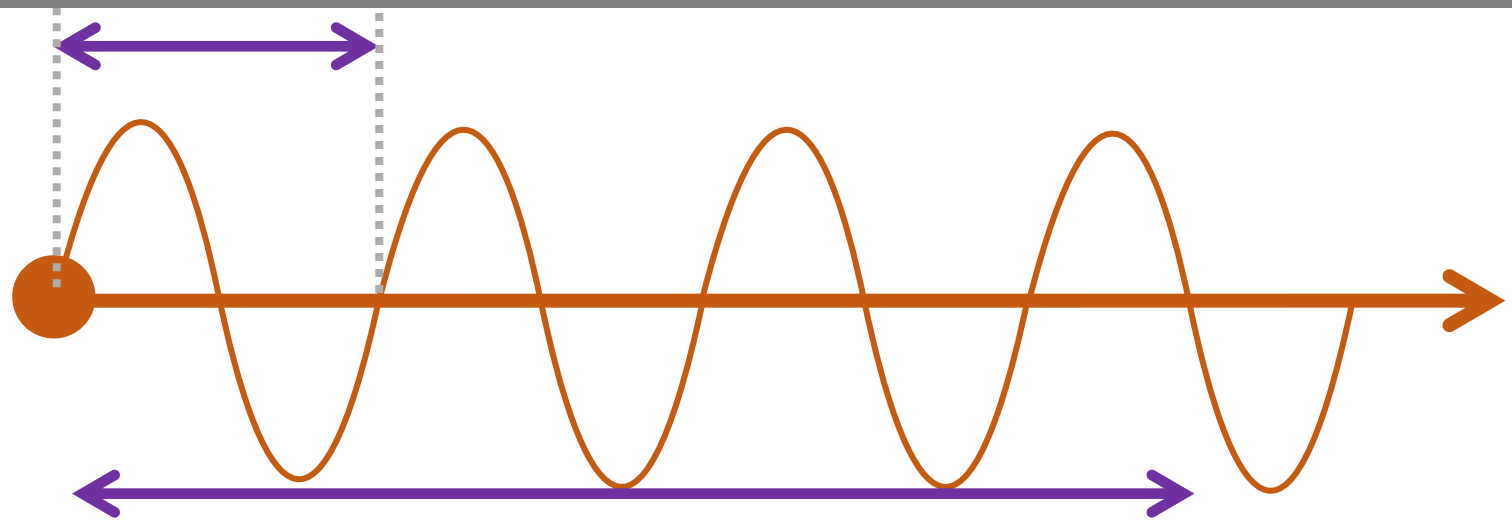
f number of cycles per second

C meters of distance per second (speed of the wave)

$distance\ per\ cycle = \lambda = C / f$

2π radians of angle is covered per cycle

Equation of waves in time and space



f number of cycles per second

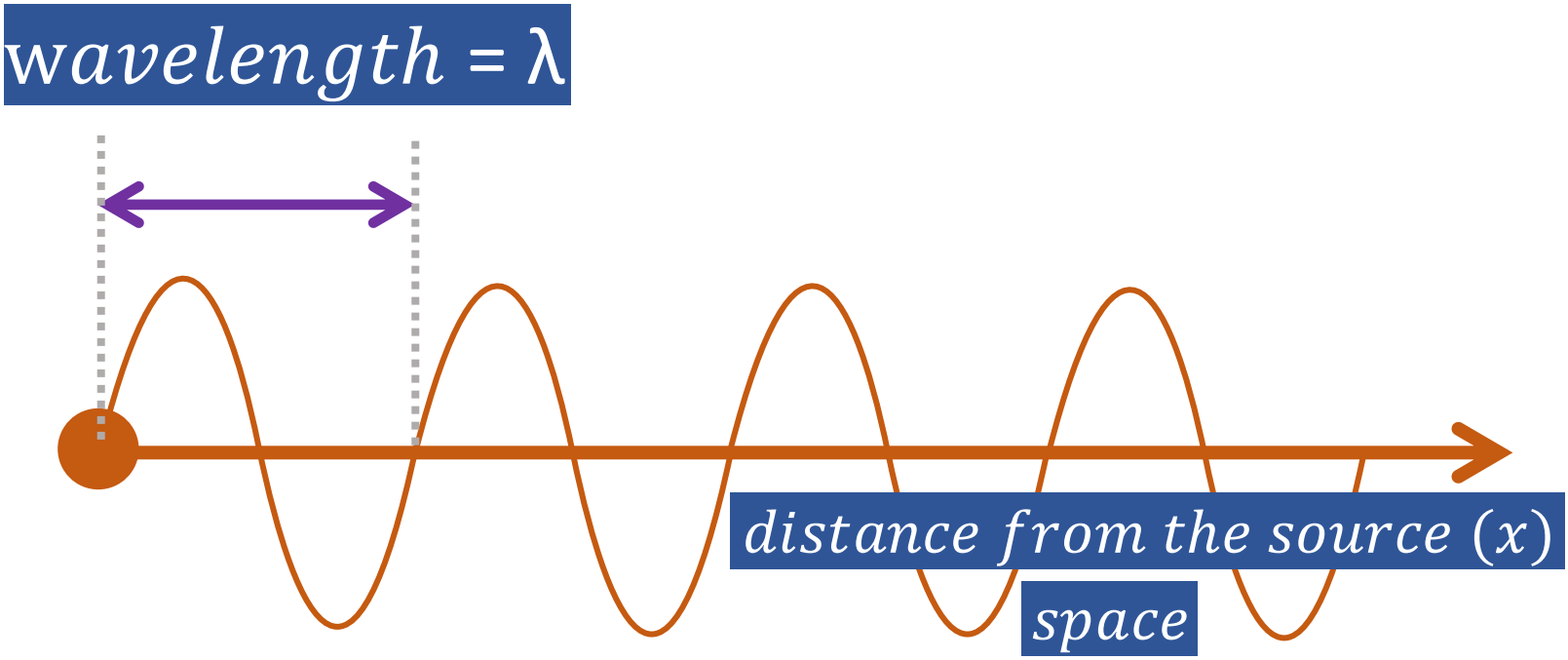
C meters of distance per second
(speed of the wave)

distance per cycle $= \lambda = C / f$

2π radians of angle is covered per cycle

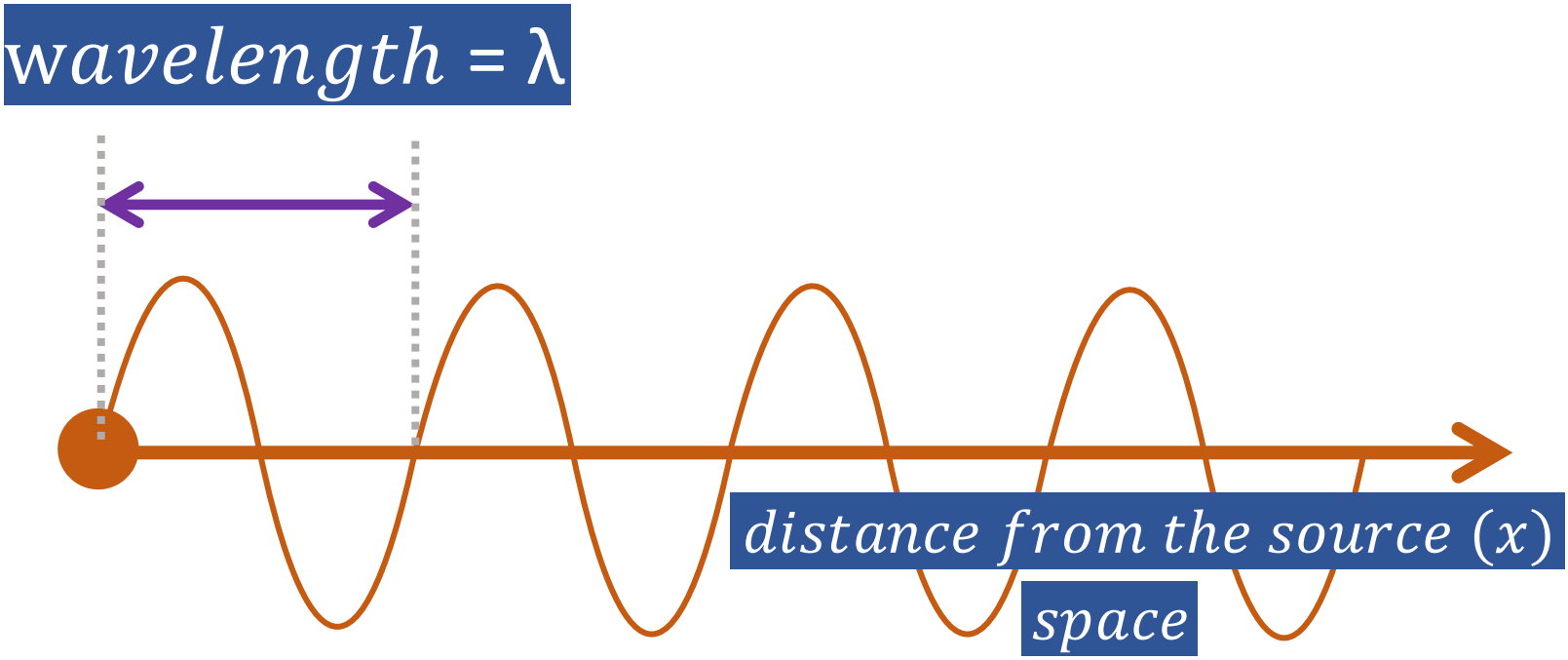
$\frac{2\pi}{\lambda}$ radians of angle is covered per unit distance

Equation of waves in time and space



$$\Psi(t, x) = A \cdot \sin(2\pi f t - \theta(x))$$

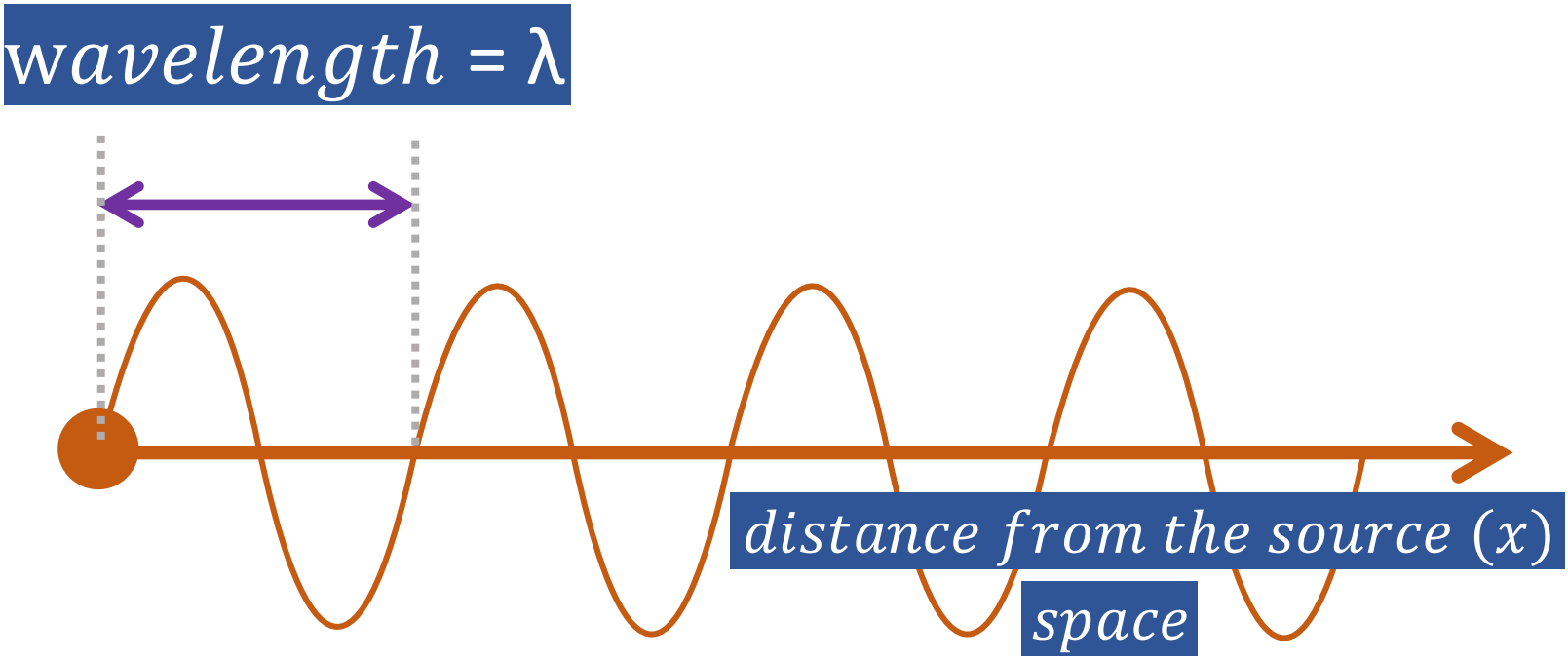
Equation of waves in time and space



$$\Psi(t, x) = A \,.\, \sin(2\pi f t - \theta(x))$$

$$= A \,.\, \sin(2\pi f t - \frac{2\pi}{\lambda} x)$$

Equation of waves in time and space



$$\Psi(t, x) = A \,.\, \sin(2\pi f t - \theta(x))$$

$$= A \,.\, \sin(2\pi f t - \frac{2\pi}{\lambda} x)$$

$$= A \,.\, e^{j(2\pi f t - \frac{2\pi}{\lambda} x)}$$

Time and space

Cycles per sec = frequency = f Hz

Distance per cycle = wavelength = λ meters

Distance per second = speed = C meters/sec

$$C = f \cdot \lambda$$

Speed of sound in air: 343 m/s

Speed of sound in water: 1493 m/s

Speed of sound in iron: 5130 m/s

Speed of electromagnetic waves: $3*10^8$ m/s
(~ a million times faster than sound)

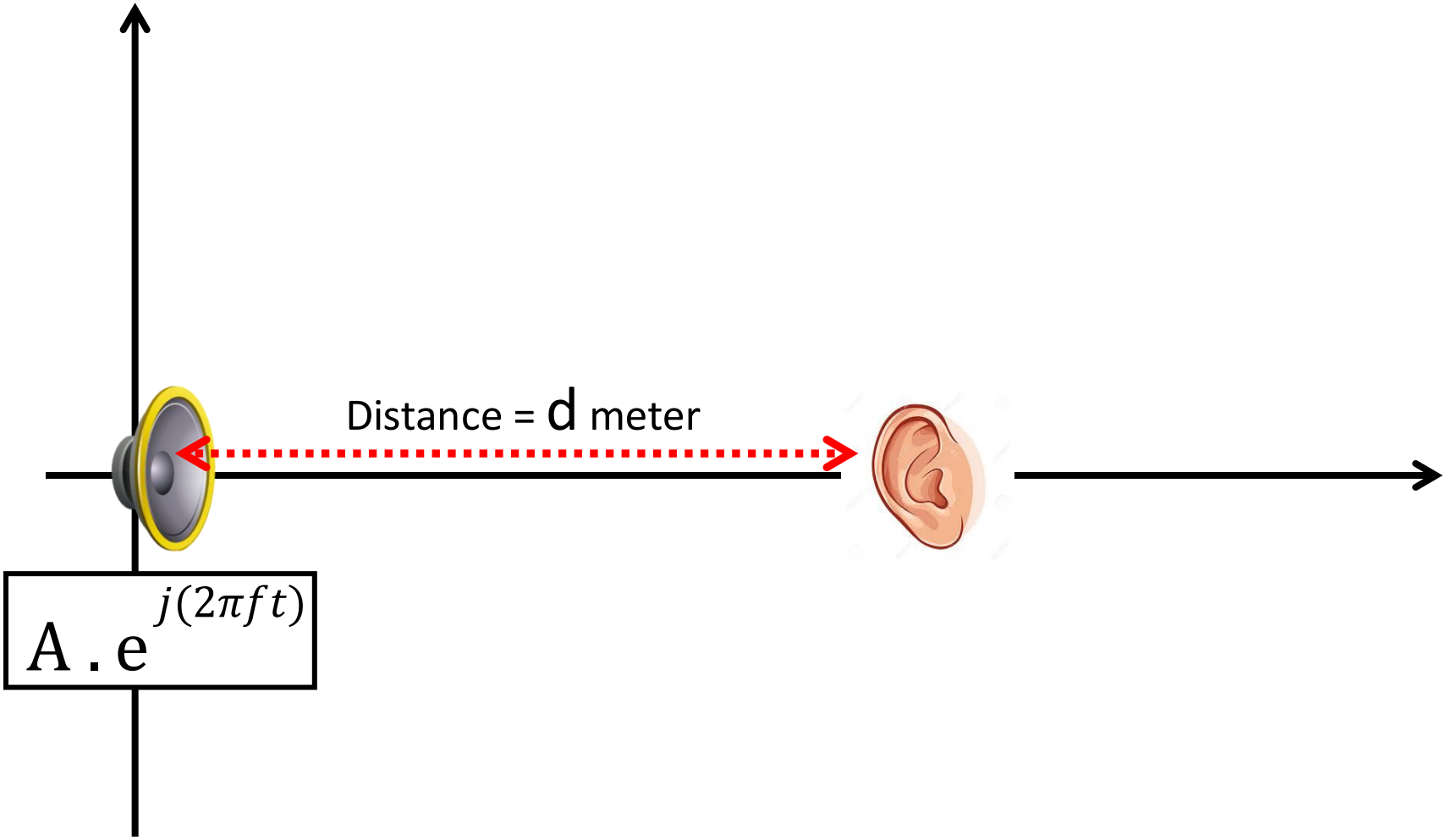
Time and space



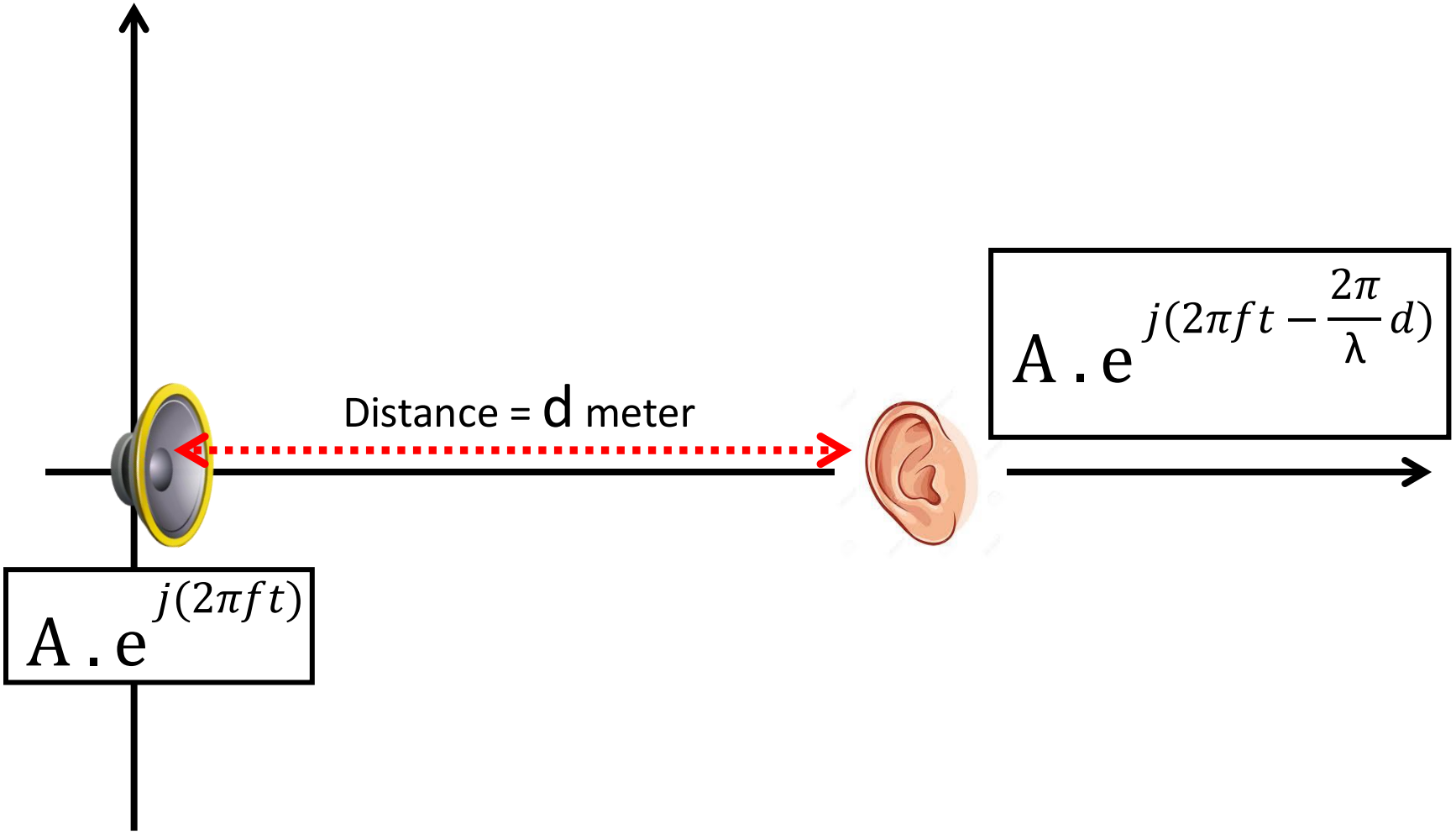
Time and space



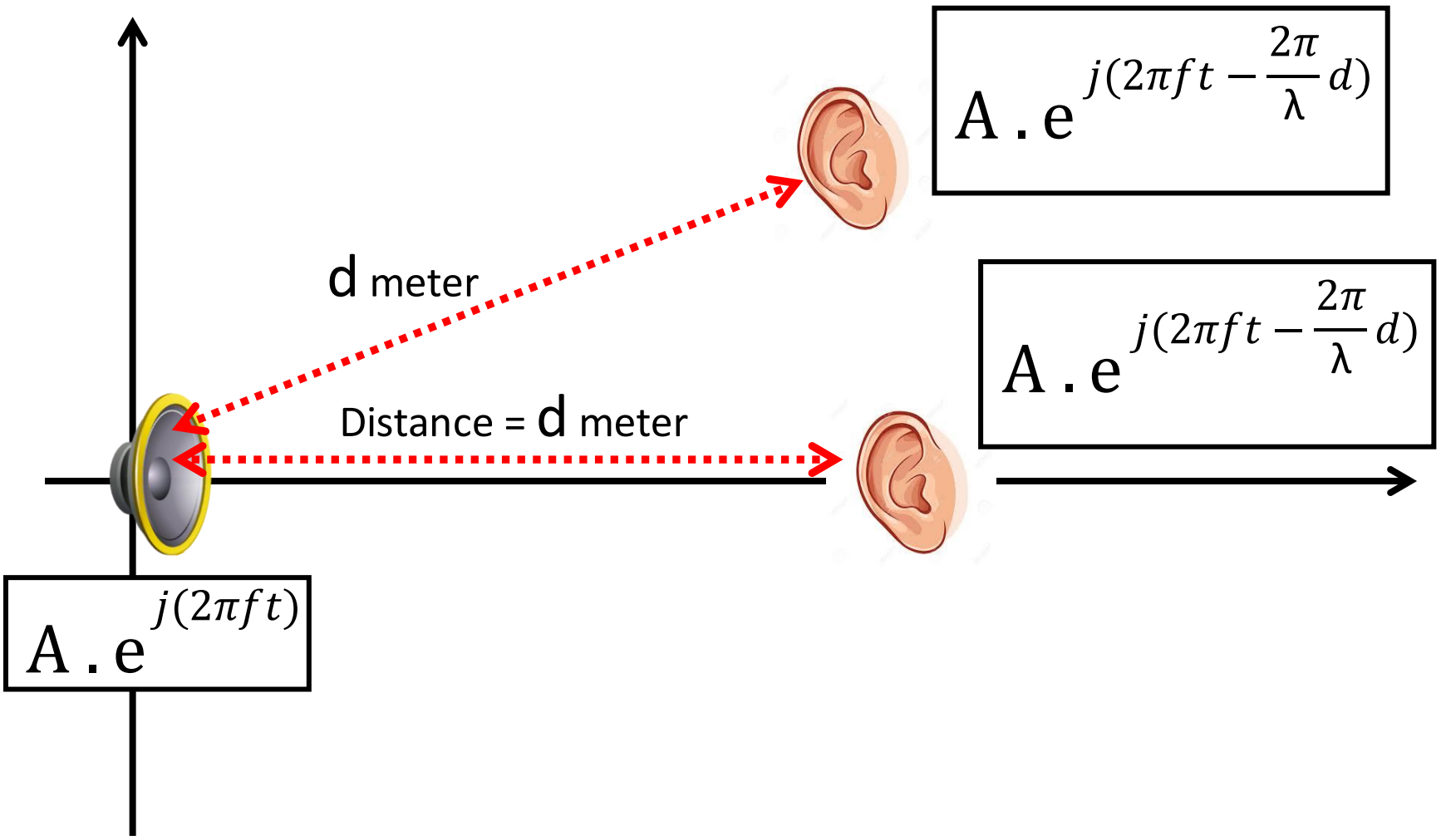
Time and space



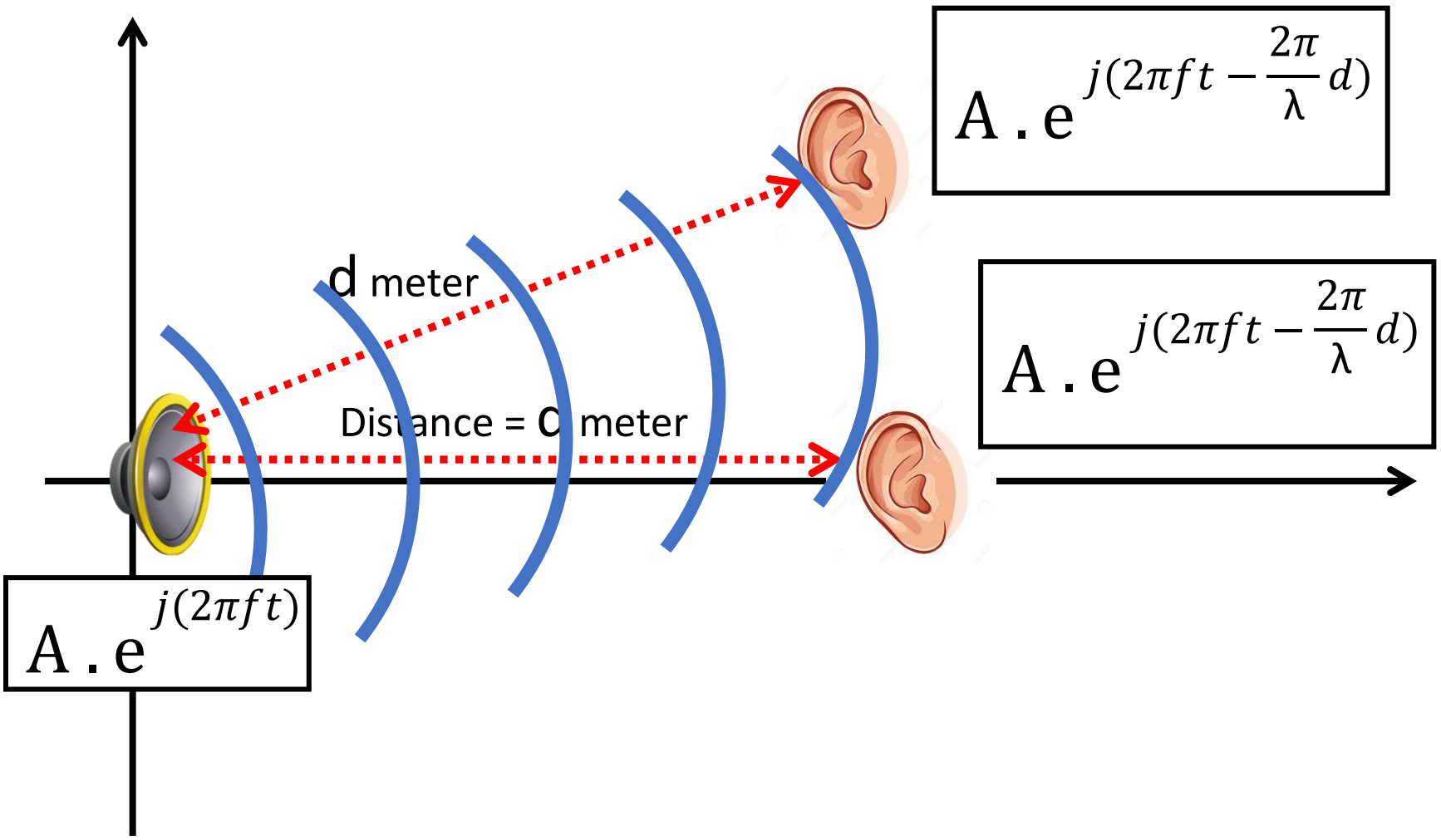
Time and space



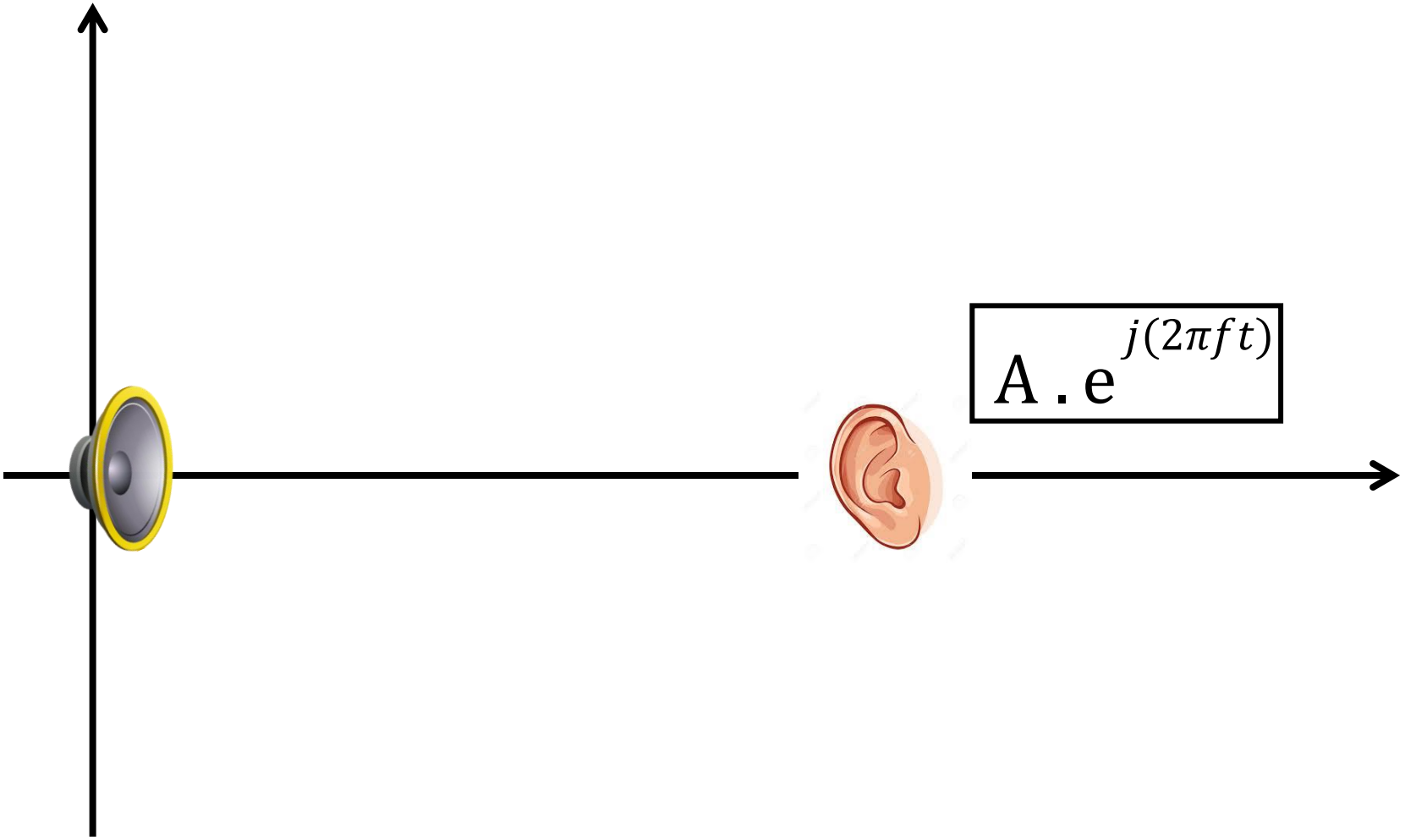
Time and space



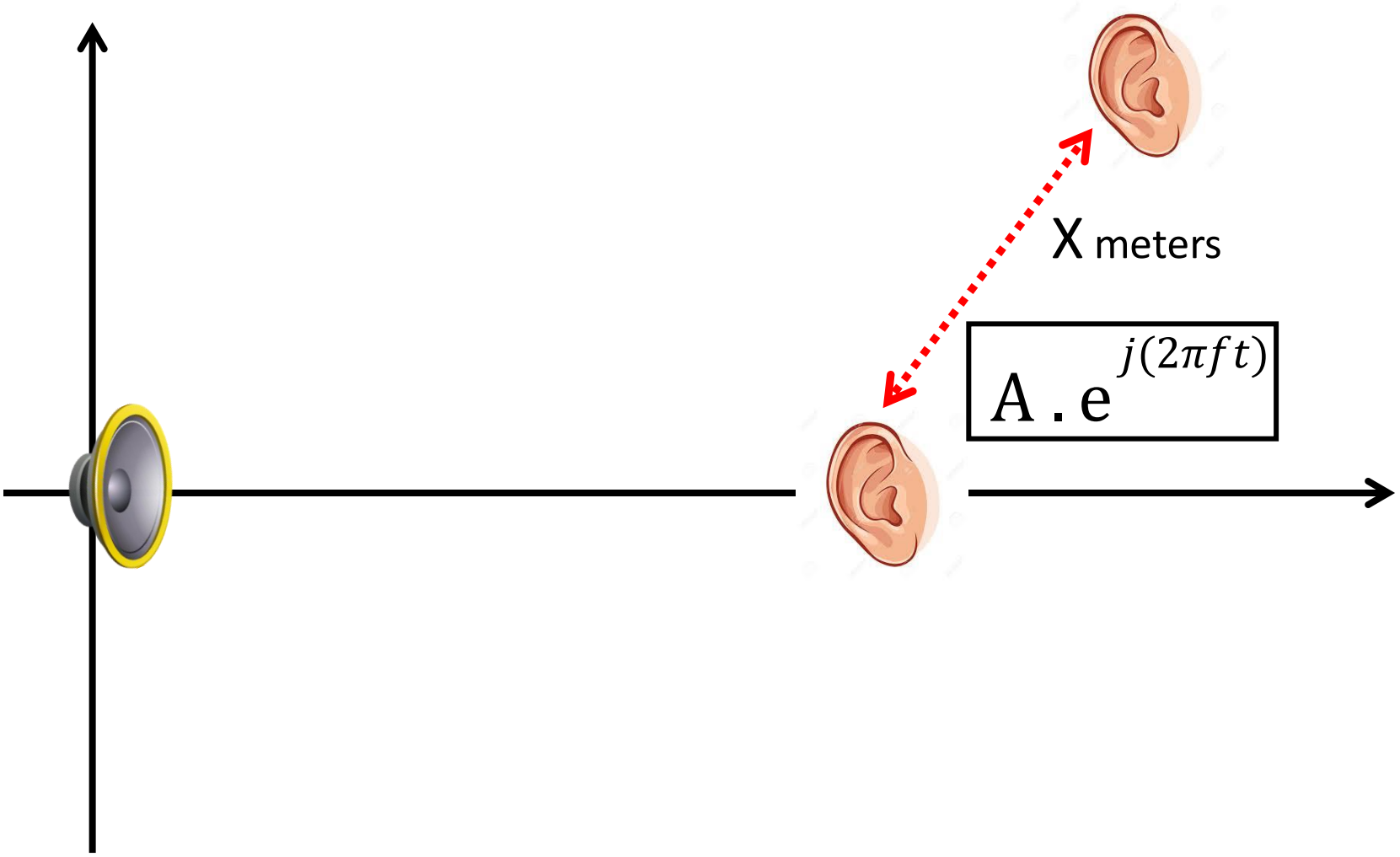
Time and space



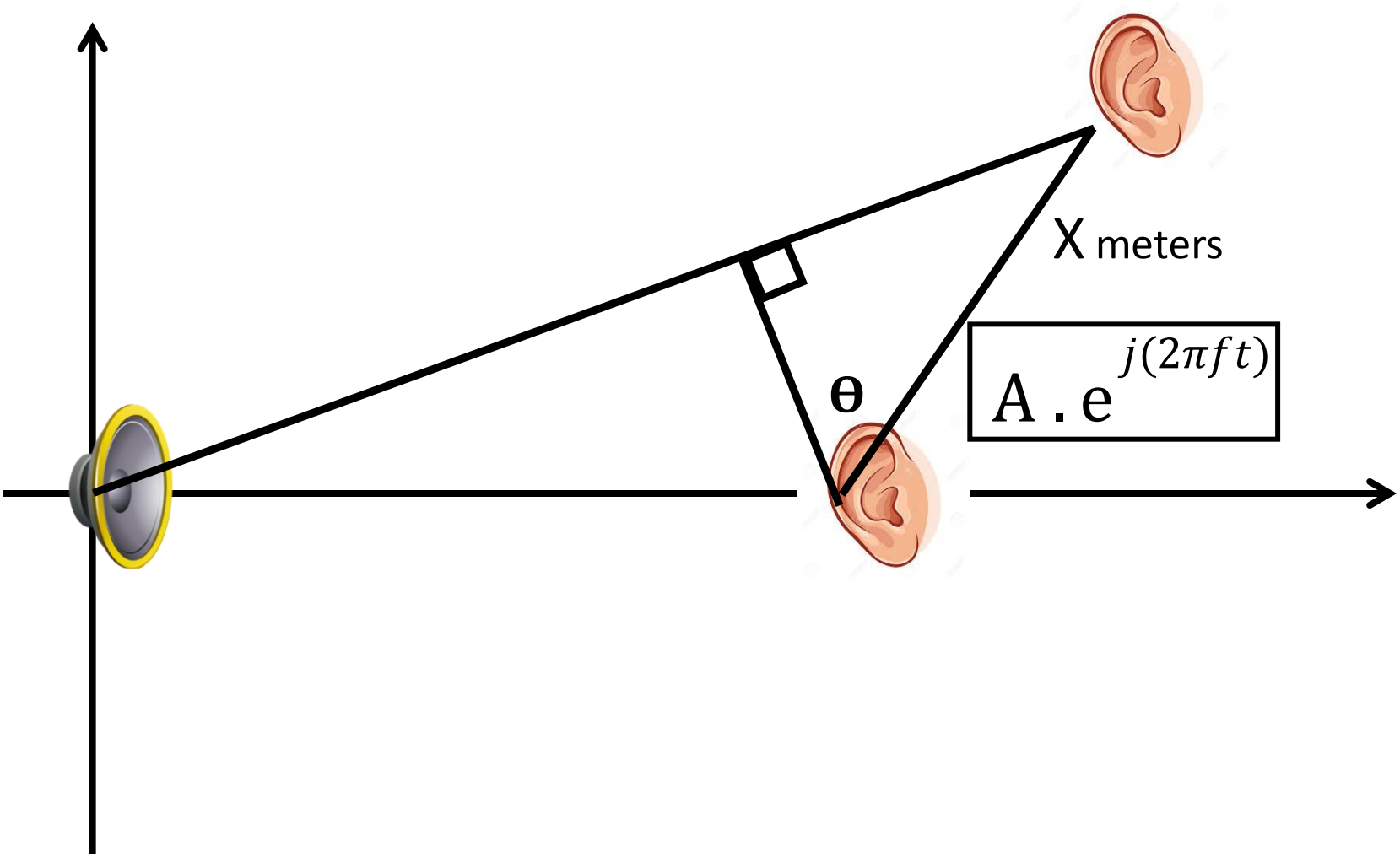
Time and space



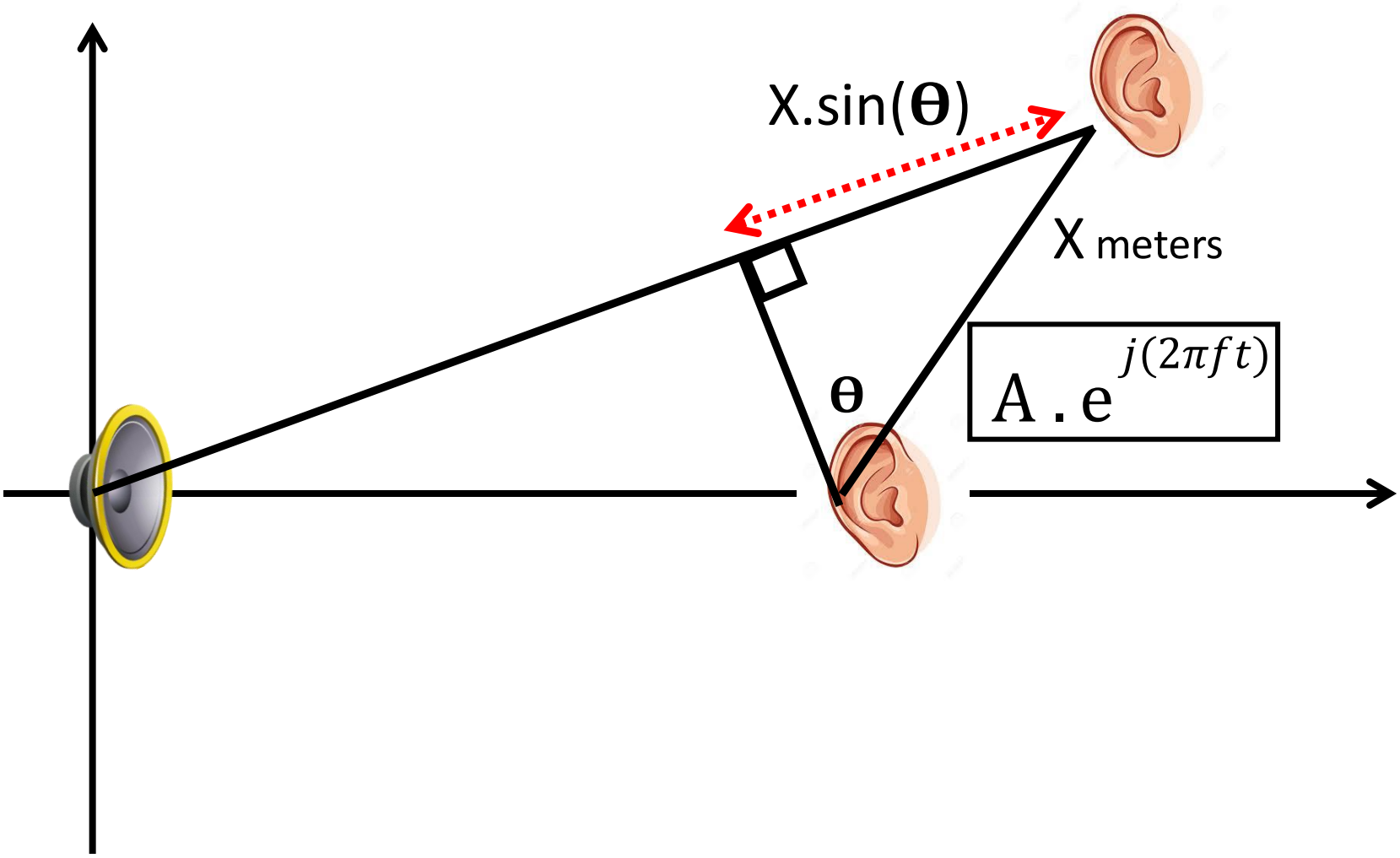
Time and space



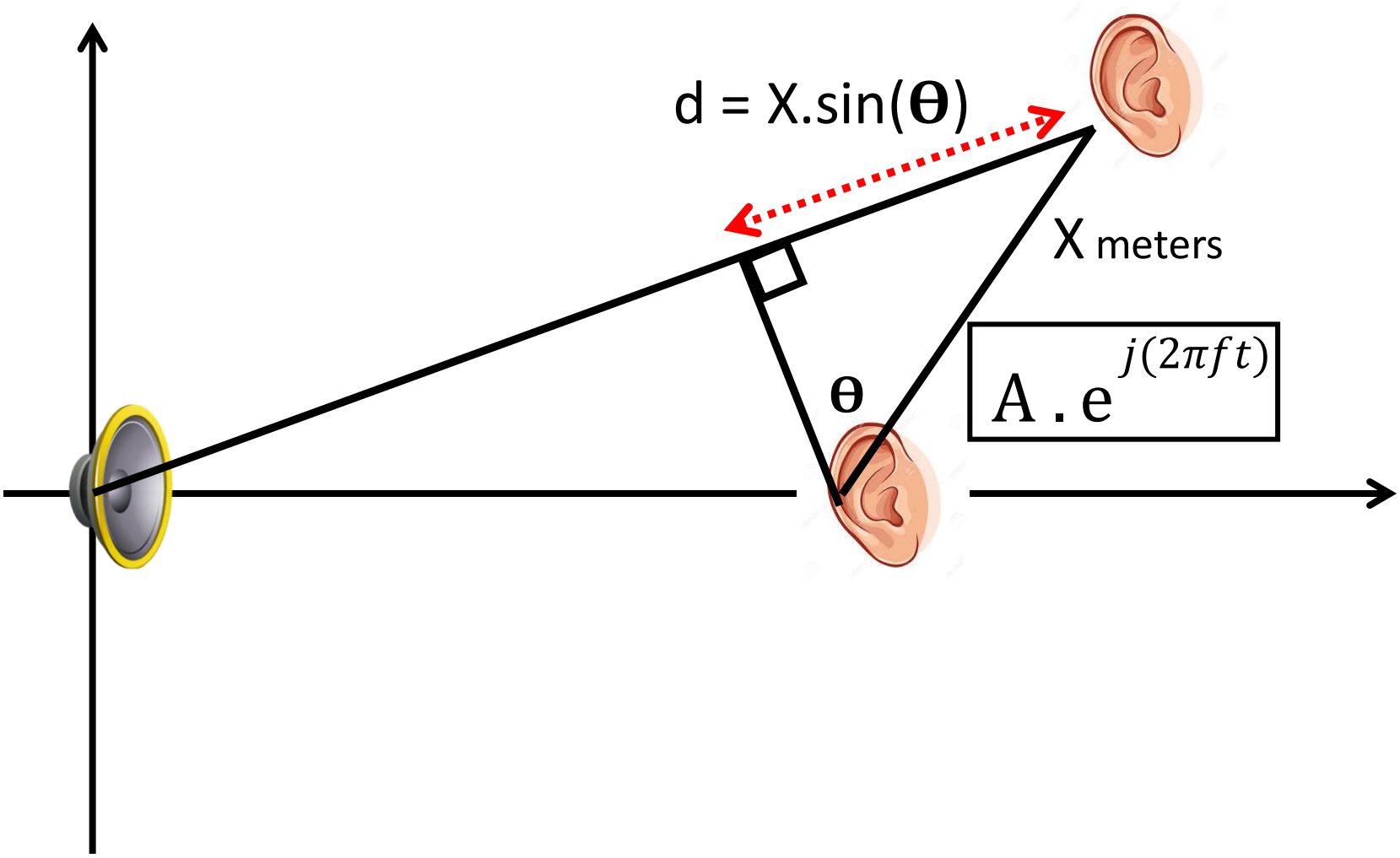
Time and space



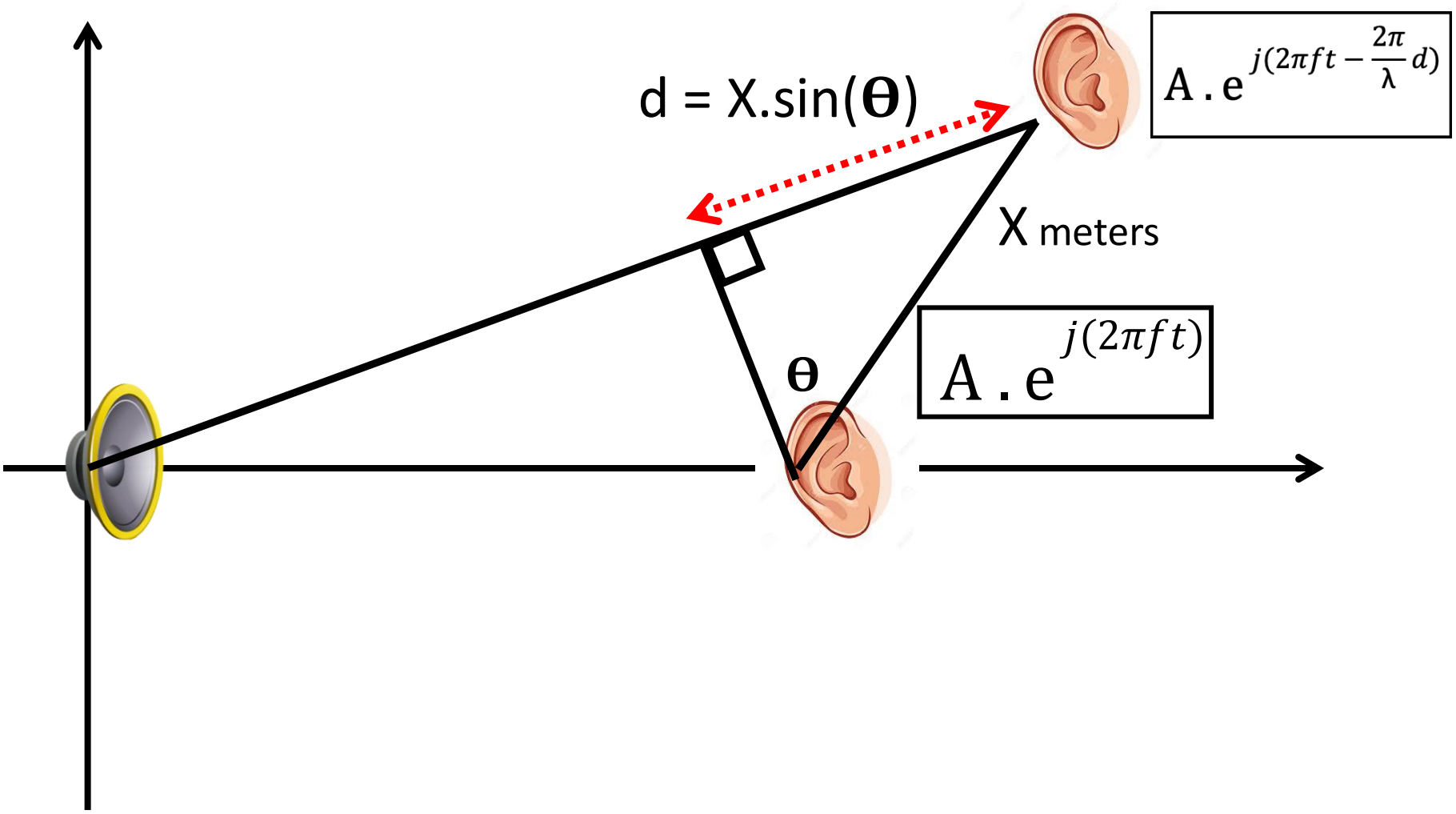
Time and space



Time and space

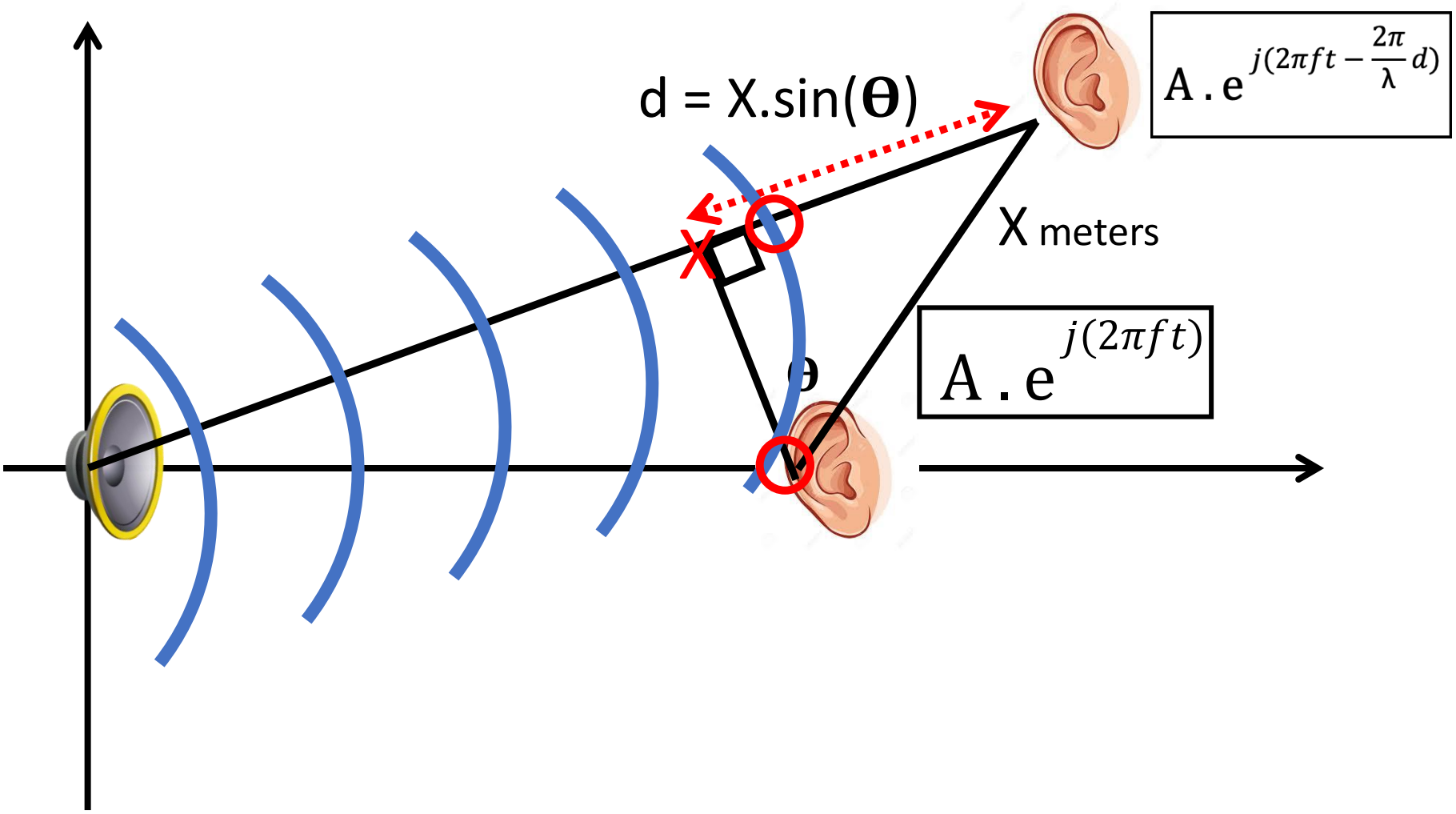


Time and space

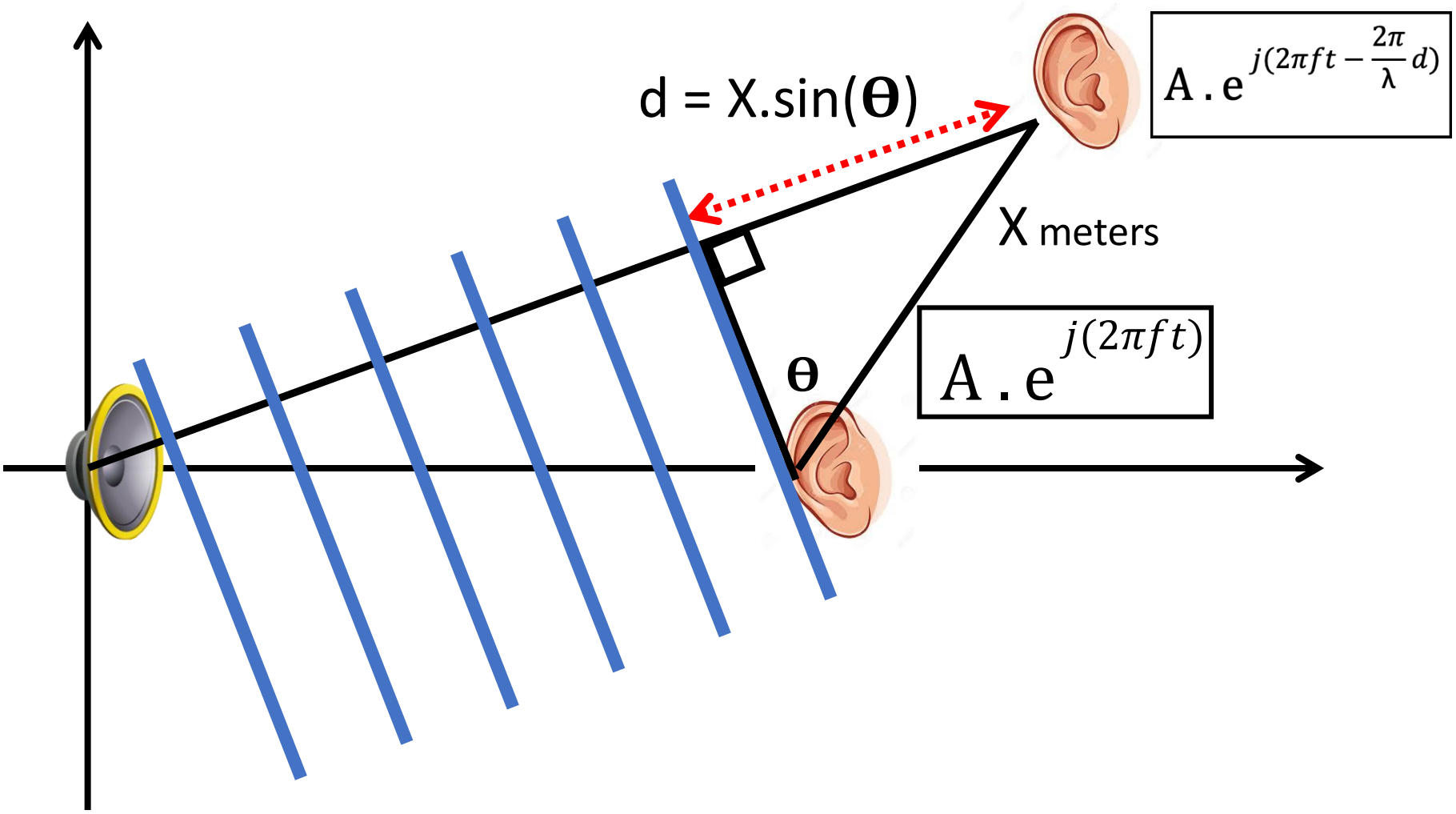


Did you notice a small error?

Time and space

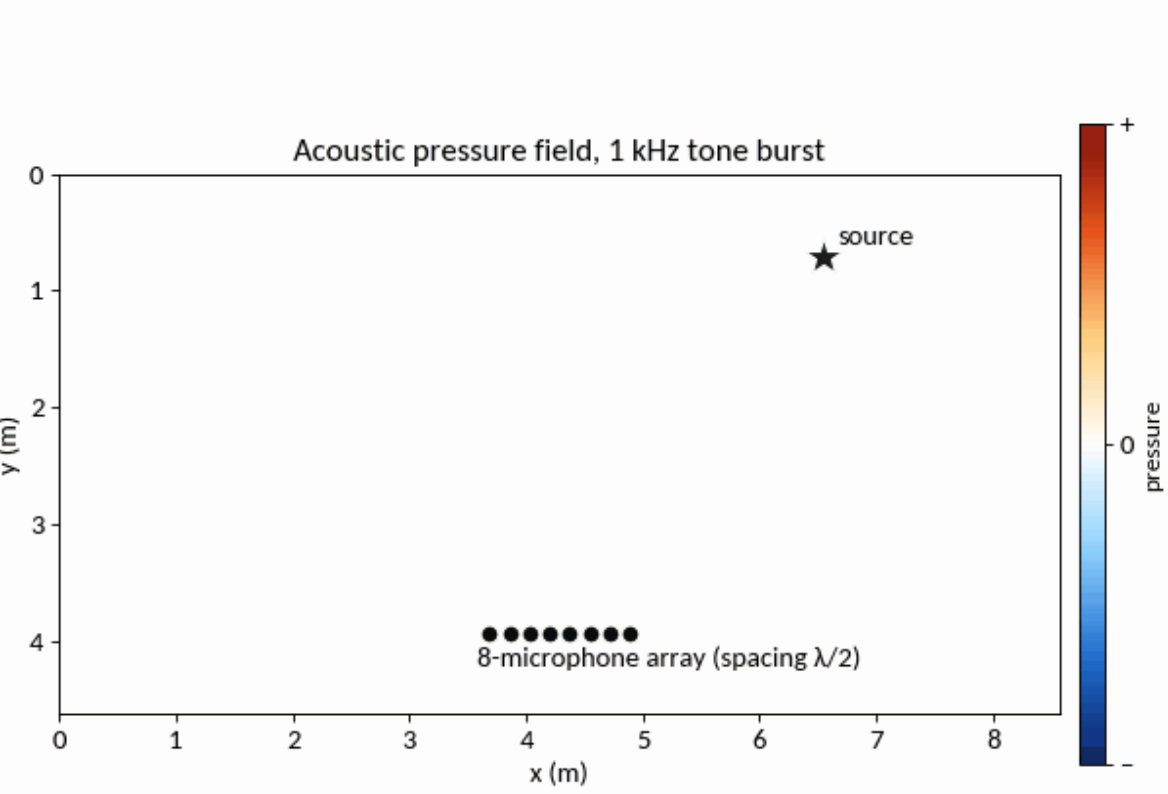


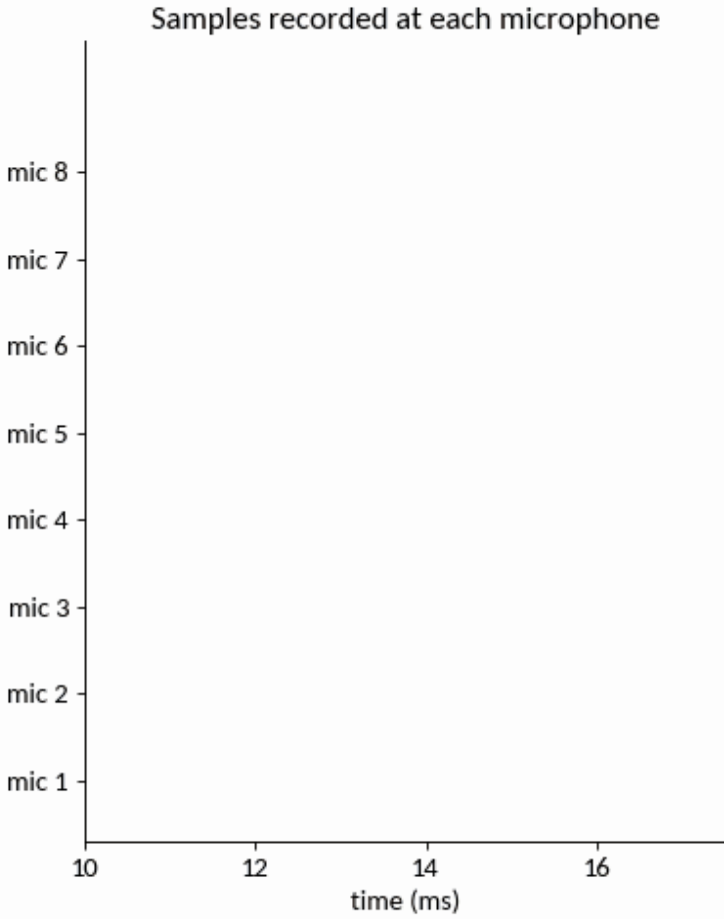
Time and space



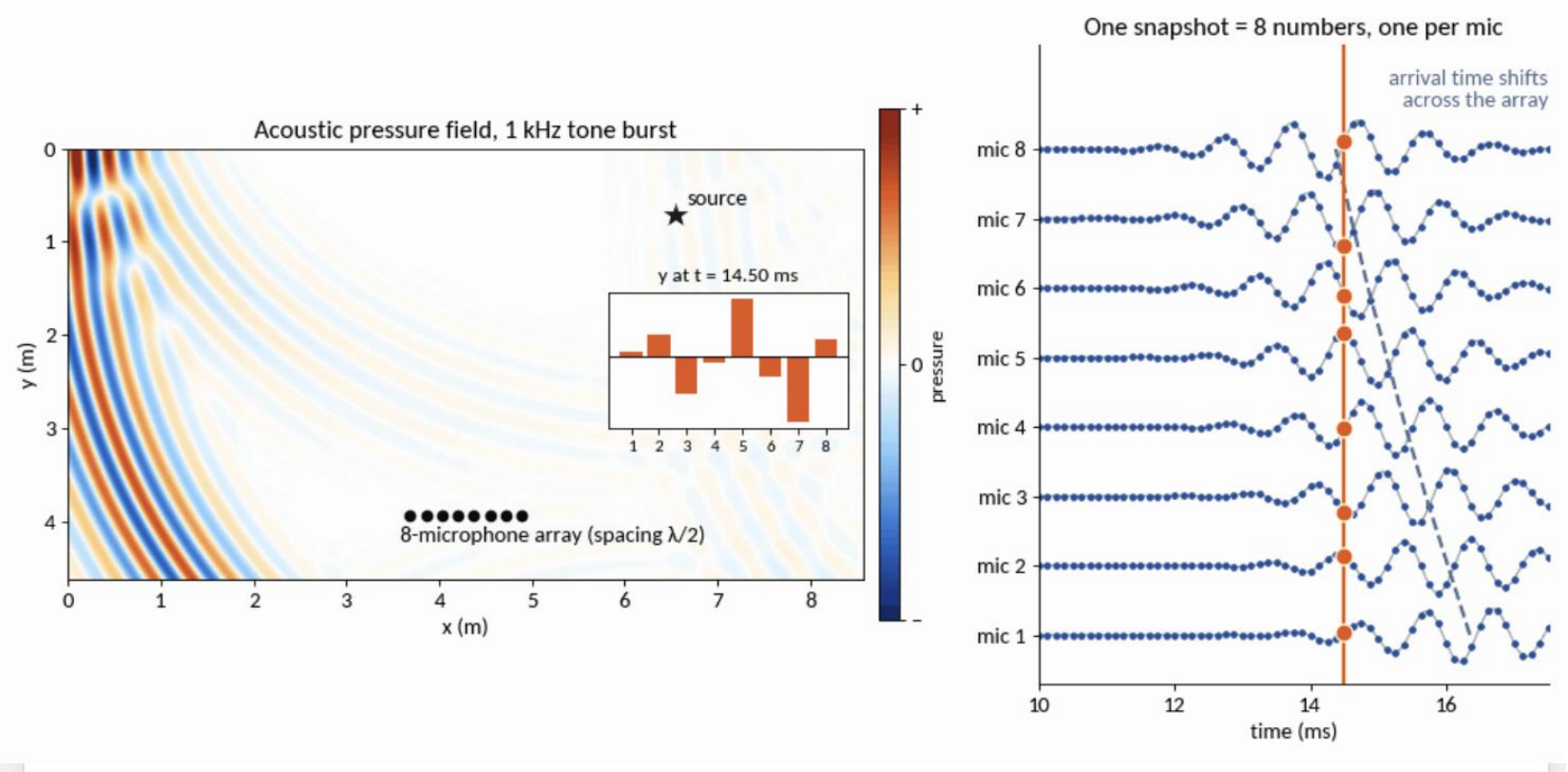
Far-field approximation, Planar wave

Spatial Sampling using an Array of Sensors





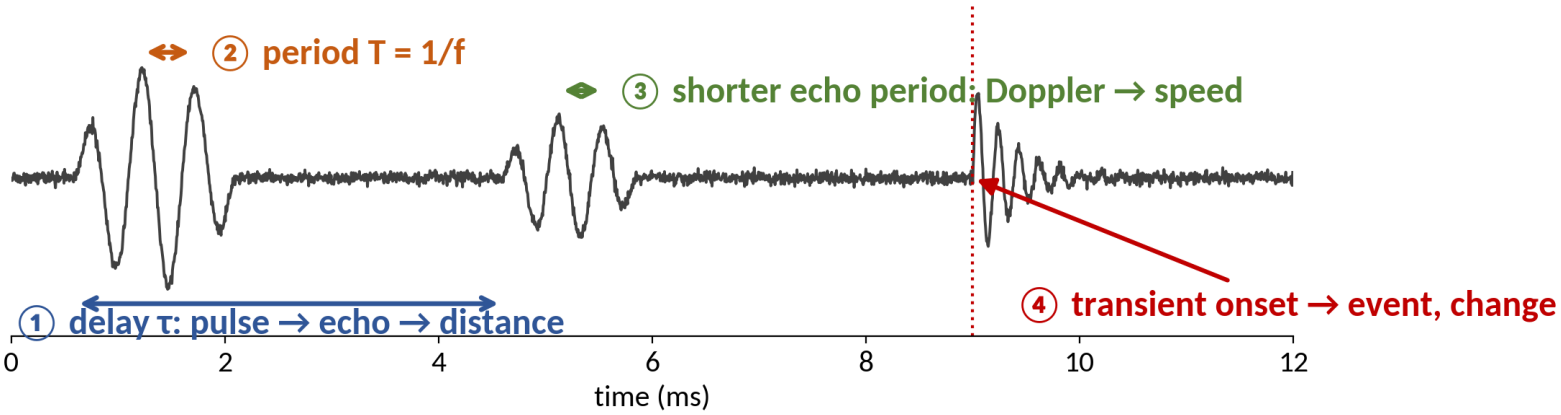
Spatial Sampling using an Array of Sensors



Temporal Sensing: What Can a Signal Reveal Over Time?

Time evolution → events, dynamics, frequency, and motion

ONE SENSOR, ONE RECORDING y(t)



WHAT IT REVEALS

① **Timing / delay:** ToA, TDoA

② **Frequency:** period, spectrum

Phase / correlation: alignment, sync

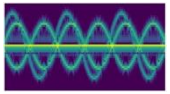
③ **Doppler / rate:** speed

④ **Dynamics:** transients, state changes

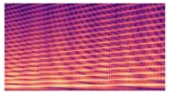
WHERE IT IS USED



Ranging
echo / time of flight, localization, sync



Motion
Doppler speed, gestures, activity



Audio / speech
pitch, rhythm, event detection, recognition



Biomedical
heart rhythm, gait, physiological dynamics



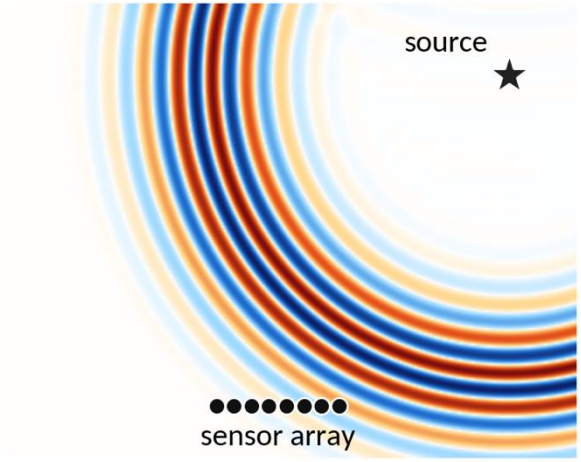
Machines / RF
fault transients, spectrum sensing, sync

A signal in time reveals when things happen, how fast, and what caused them.

Spatial Sensing: What Can an Array Reveal?

Space sampled at multiple locations → spatial structure

THE PHYSICS



Every sensor sees the same wave, but at a different place.

phase · delay · amplitude · correlation

(simulated 2-D sound field)

WHAT IT REVEALS

Direction / angle
DoA, azimuth, elevation

Range / position
distance, 2-D / 3-D location

Motion
velocity, trajectory, pose

Multiplicity
how many sources / targets

Spatial structure
shape, reflectors, images

WHERE IT IS USED



Acoustics
find & separate talkers, beamforming



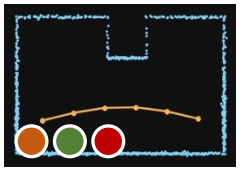
Wireless / RF
device localization, multipath maps



Radar
angle + range, tracking, imaging



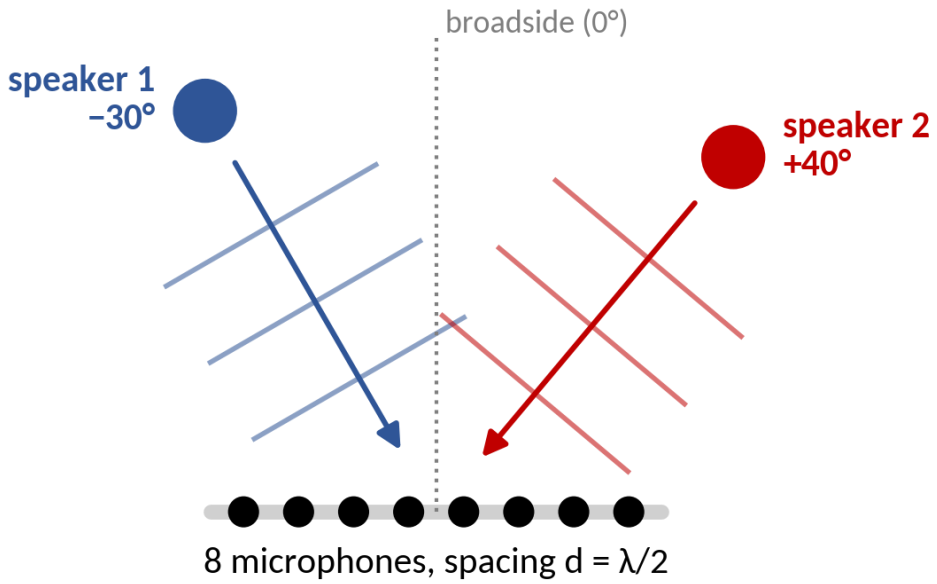
Sonar
underwater localization, imaging



Robotics / XR
SLAM, pose tracking, mapping

Multiple spatial samples turn wave differences into geometry and scenes.

Where is the sound coming from?



Physical setup

- 8-microphone uniform linear array
- 1–2 far-field, narrowband sources
- one snapshot: 8 complex numbers

How can an array tell where several sources are, and what happens when classical beamforming is not enough?

Direction of Arrival is everywhere

Arrays of every shape and size

Who needs to know where a wave came from?



Smart speaker: 7-mic circle
finds the talker



Phone: 2–4 microphones
noise suppression



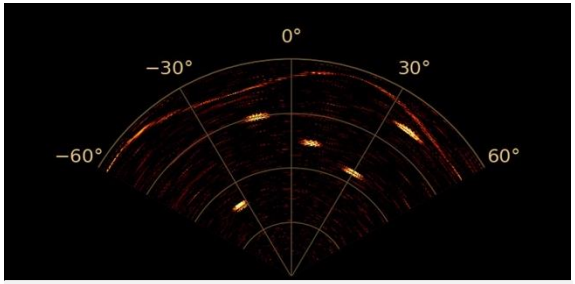
WiFi: 4-antenna array
locates devices by angle



Fighter-jet radar (AESA)
hundreds of elements



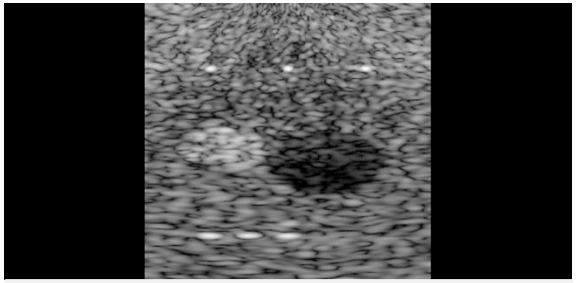
Radar operator's scope
echoes by bearing, range



Sonar: 48 hydrophones
5 targets + seabed



Handheld ultrasound
array images the heart



Ultrasound: 64 elements
cyst, lesion, point targets

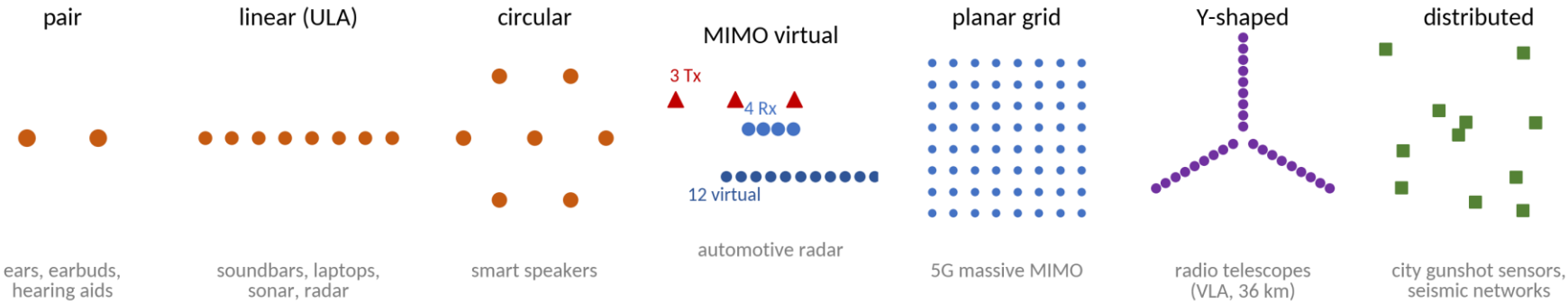


Radio telescopes (VLA)
27 dishes, one array

Sound: arrays in your pocket, ear, and living room

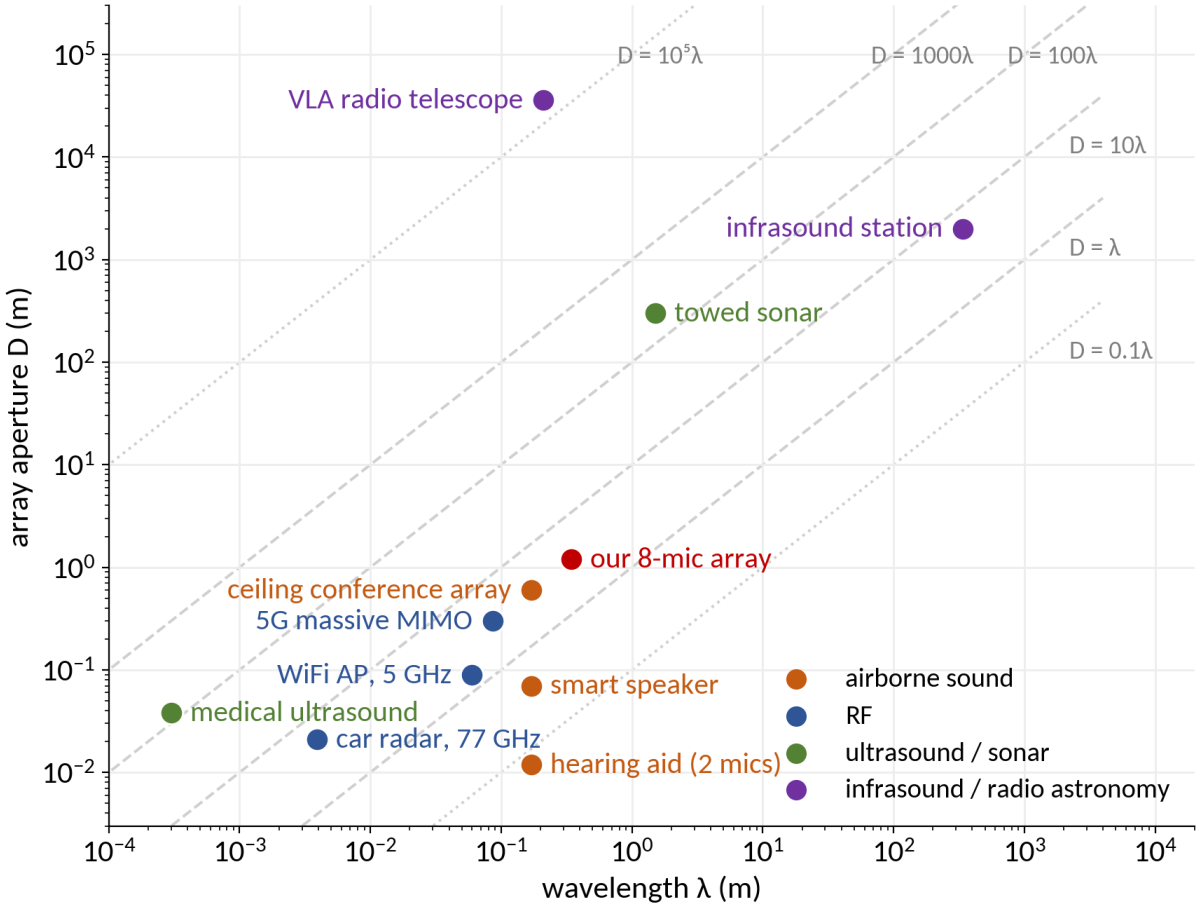
System	Array	Size	What DoA enables
Smart speaker	circle of ~6–7 mics	~7 cm	find the talker, beam toward them, reject the TV
Phone / laptop	2–4 mics	10–30 cm	noise suppression, spatial audio
Hearing aid	2 mics per ear	~1 cm	focus on the person in front
Conference ceiling	planar, dozens of mics	~0.6 m	a separate beam for each speaker
Gunshot detection	sensors spread over a city	km	locate the shot from arrival times

Arrays come in every shape



- Linear arrays see angle on one side only (front/back ambiguity).
- Circular and planar arrays see all around, or in 2-D (azimuth and elevation).
- Geometry only changes the received signal properties and sensing behavior.

…and every size: millimeters to kilometers



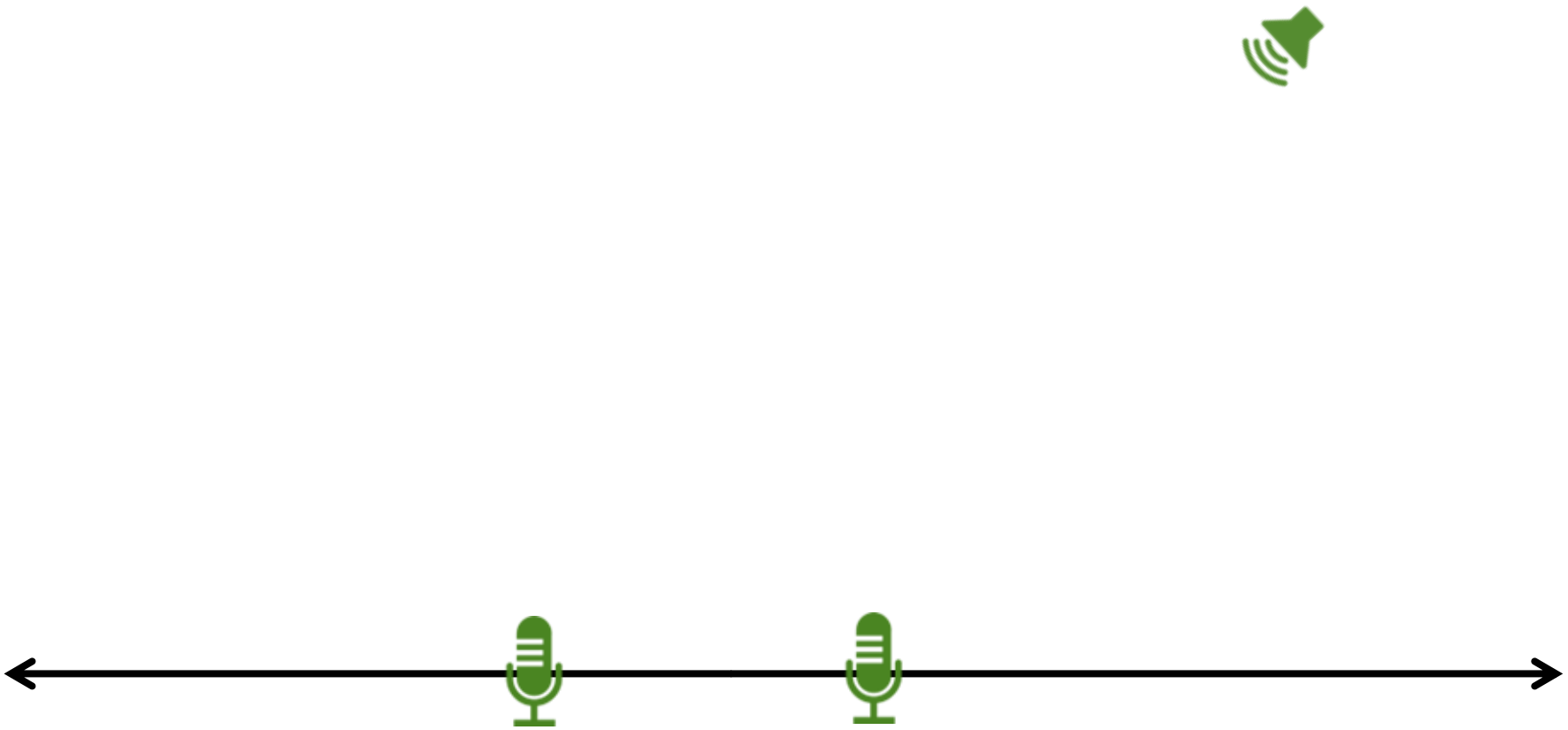
What matters is aperture measured in wavelengths, D/λ.

Angular resolution is roughly λ/D radians.

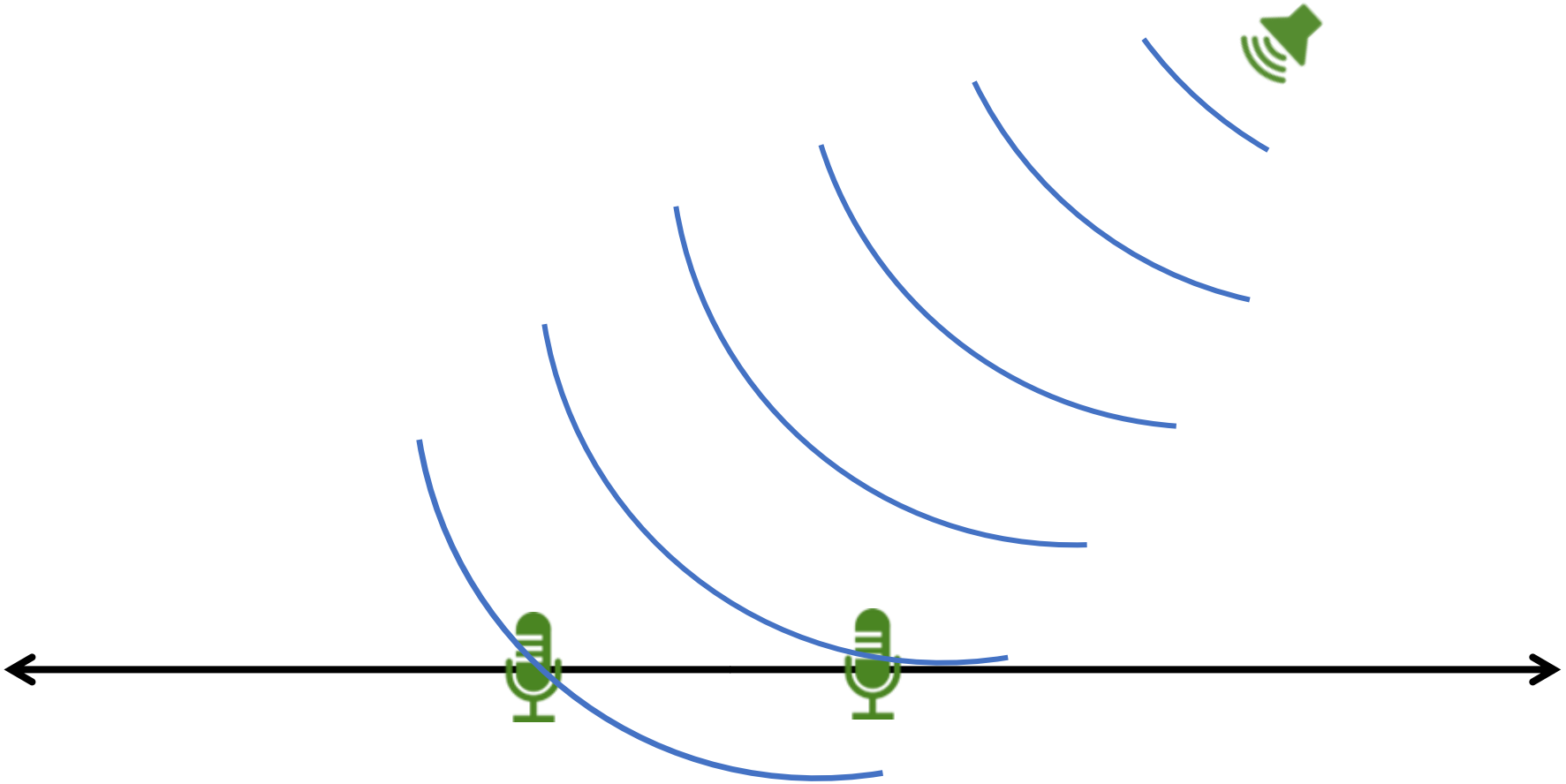
A 2 cm radar and a 1.2 m microphone array have similar D/λ, so they blur angles in similar ways.

Finite aperture limits resolution: the reason we will need priors.

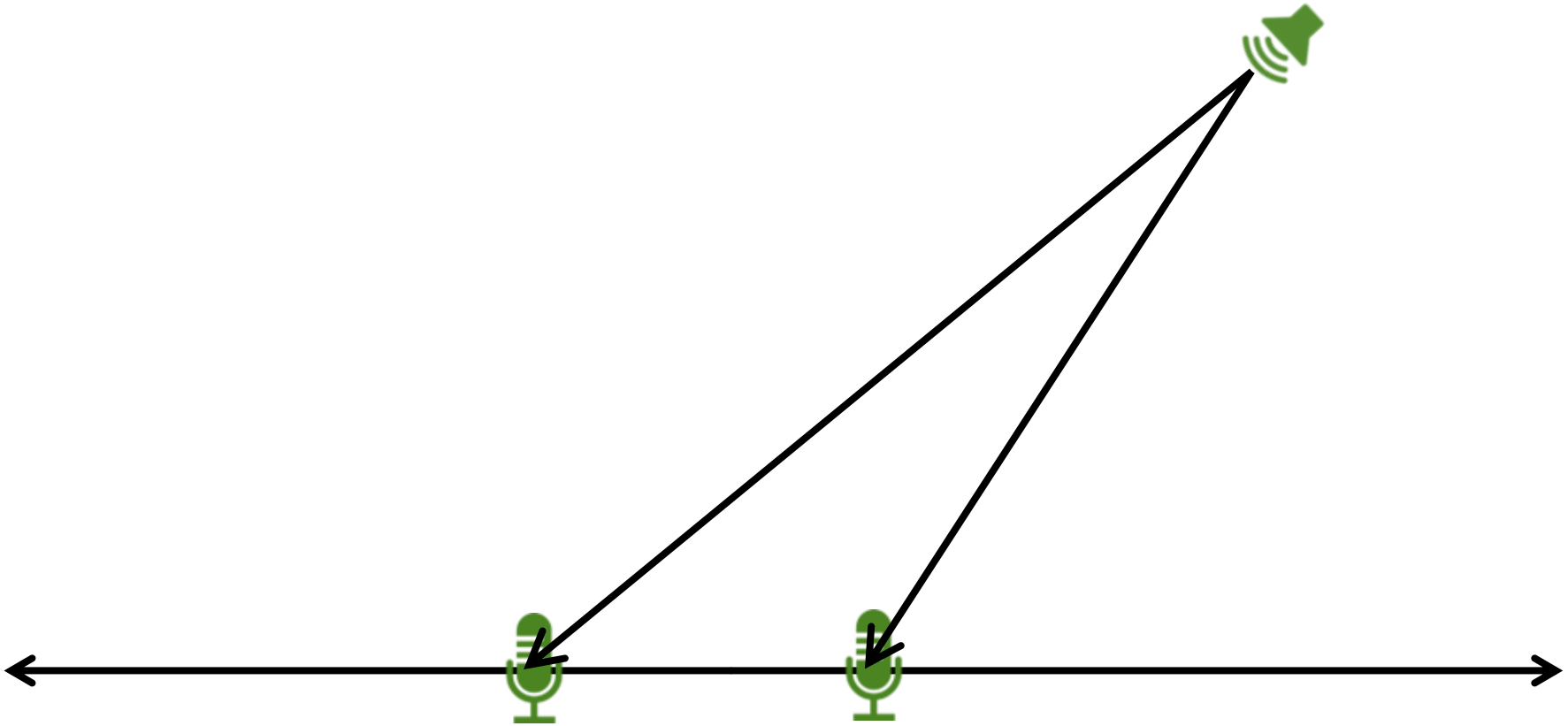
Spatial Sensing with an Array



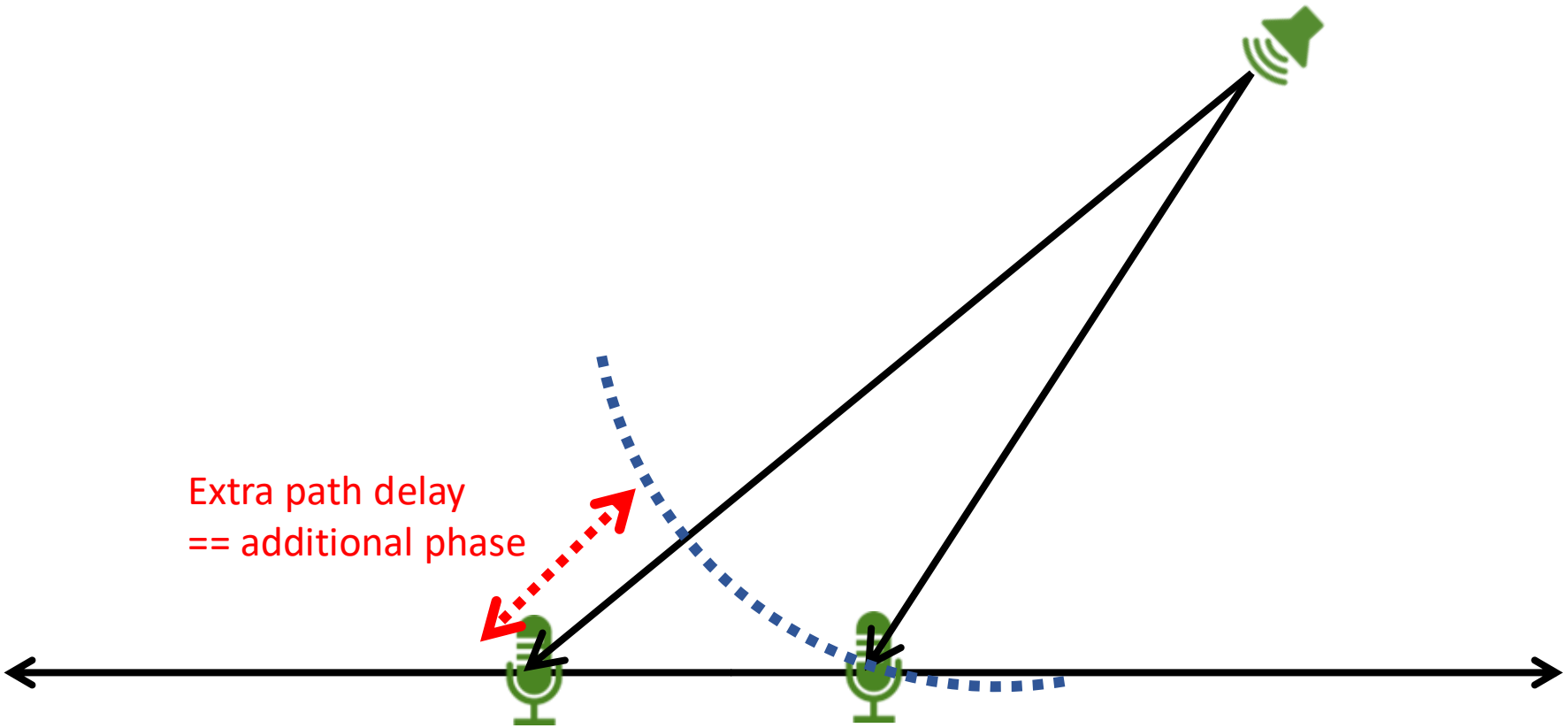
Spatial Sensing with an Array



Spatial Sensing with an Array

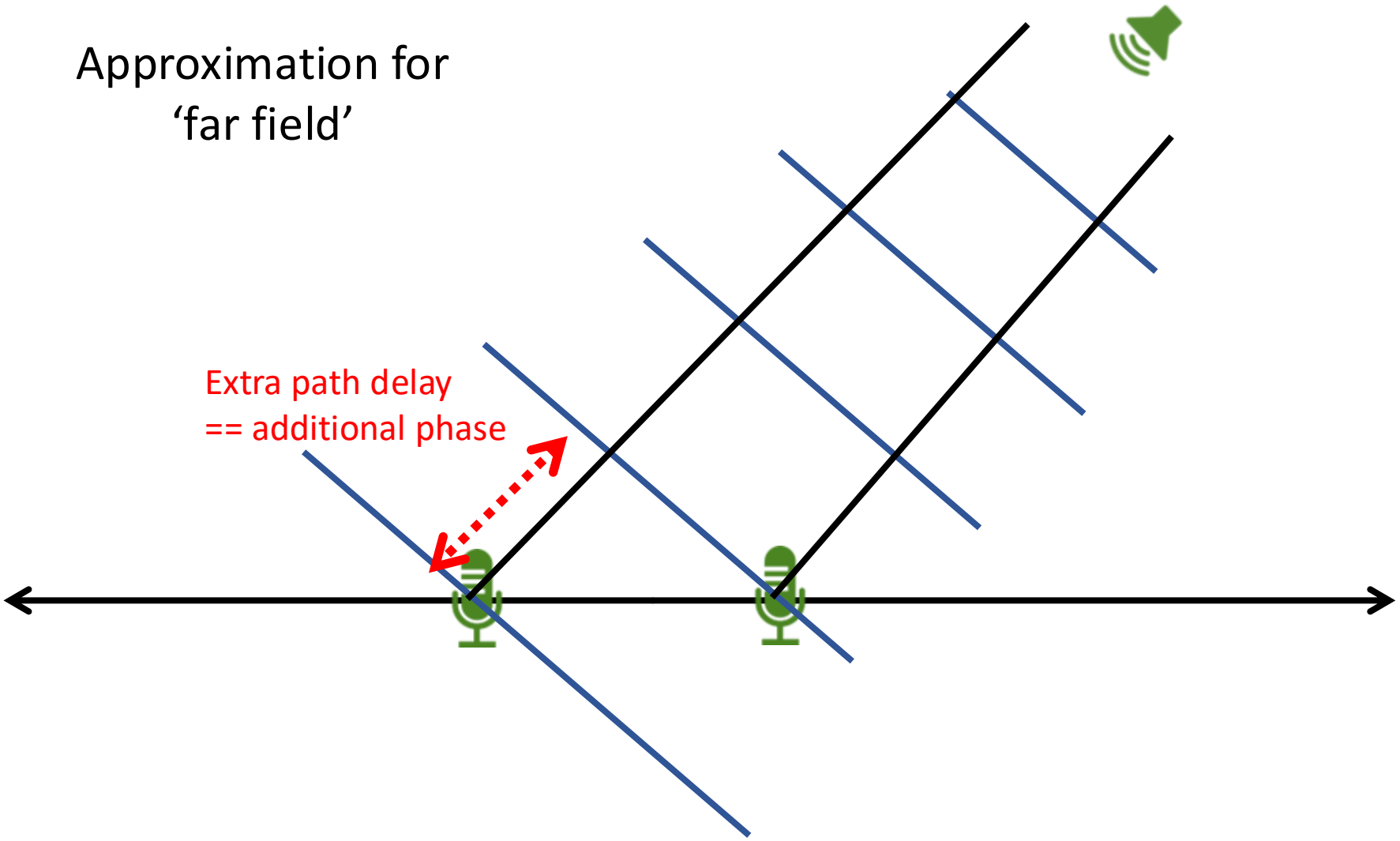


Spatial Sensing with an Array



Spatial Sensing with an Array

Approximation for
'far field'



Spatial Sensing with an Array

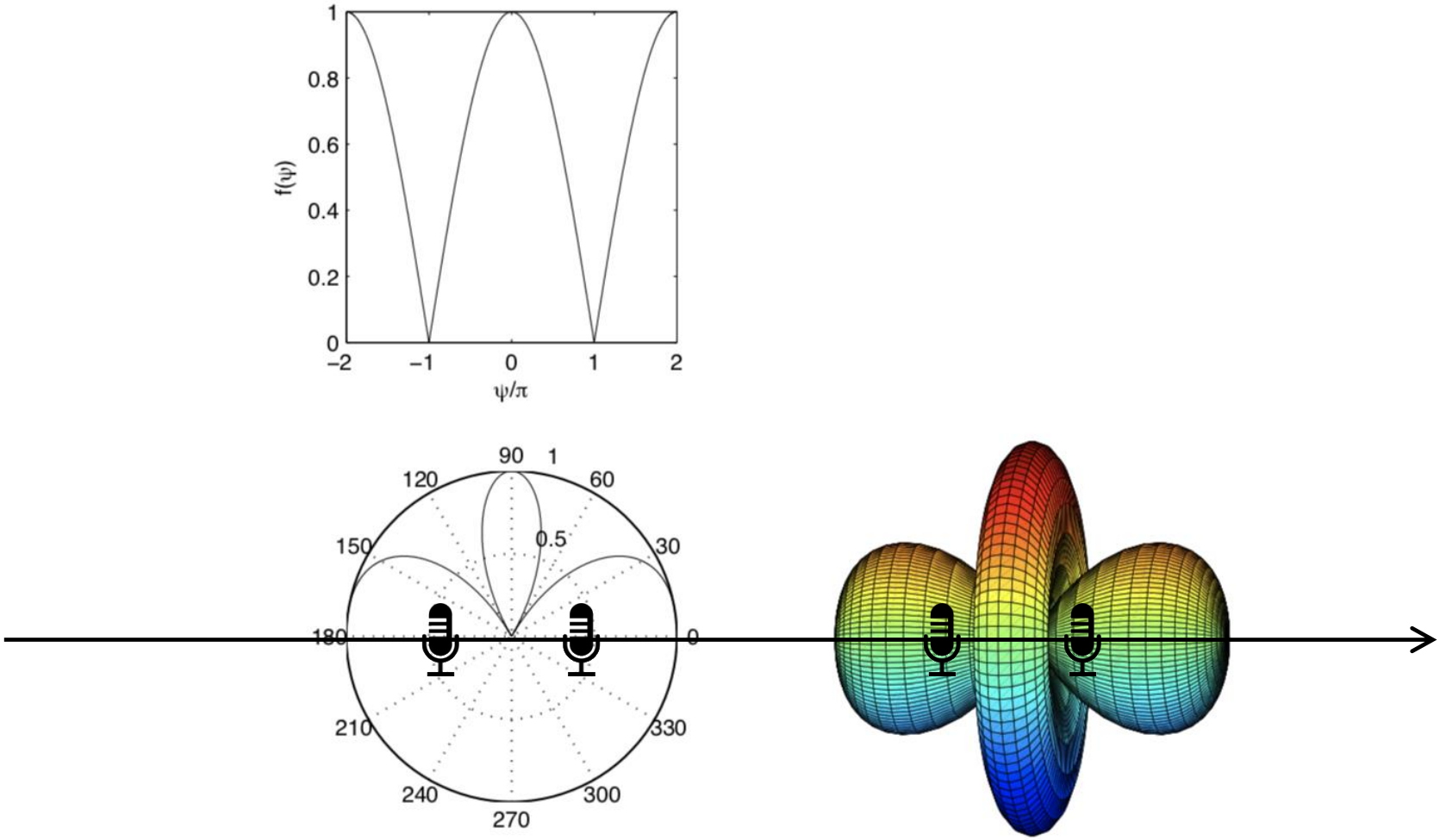
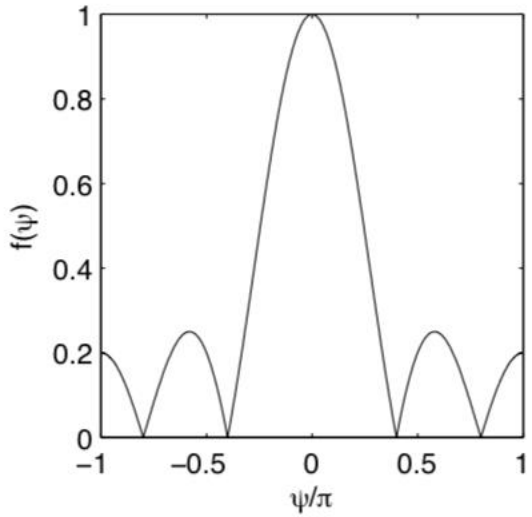


Figure 5: Pattern construction for $N = 2$, $d = \lambda$

Spatial Sensing with an Array



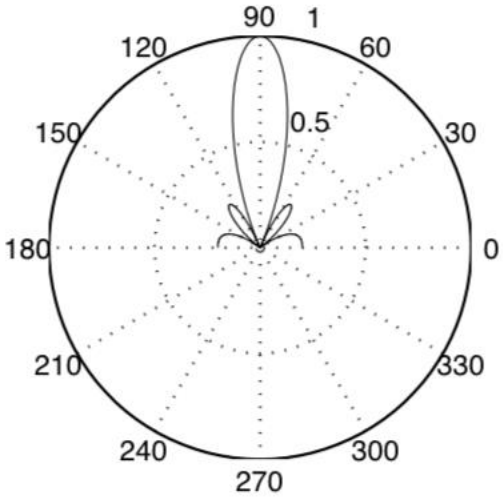
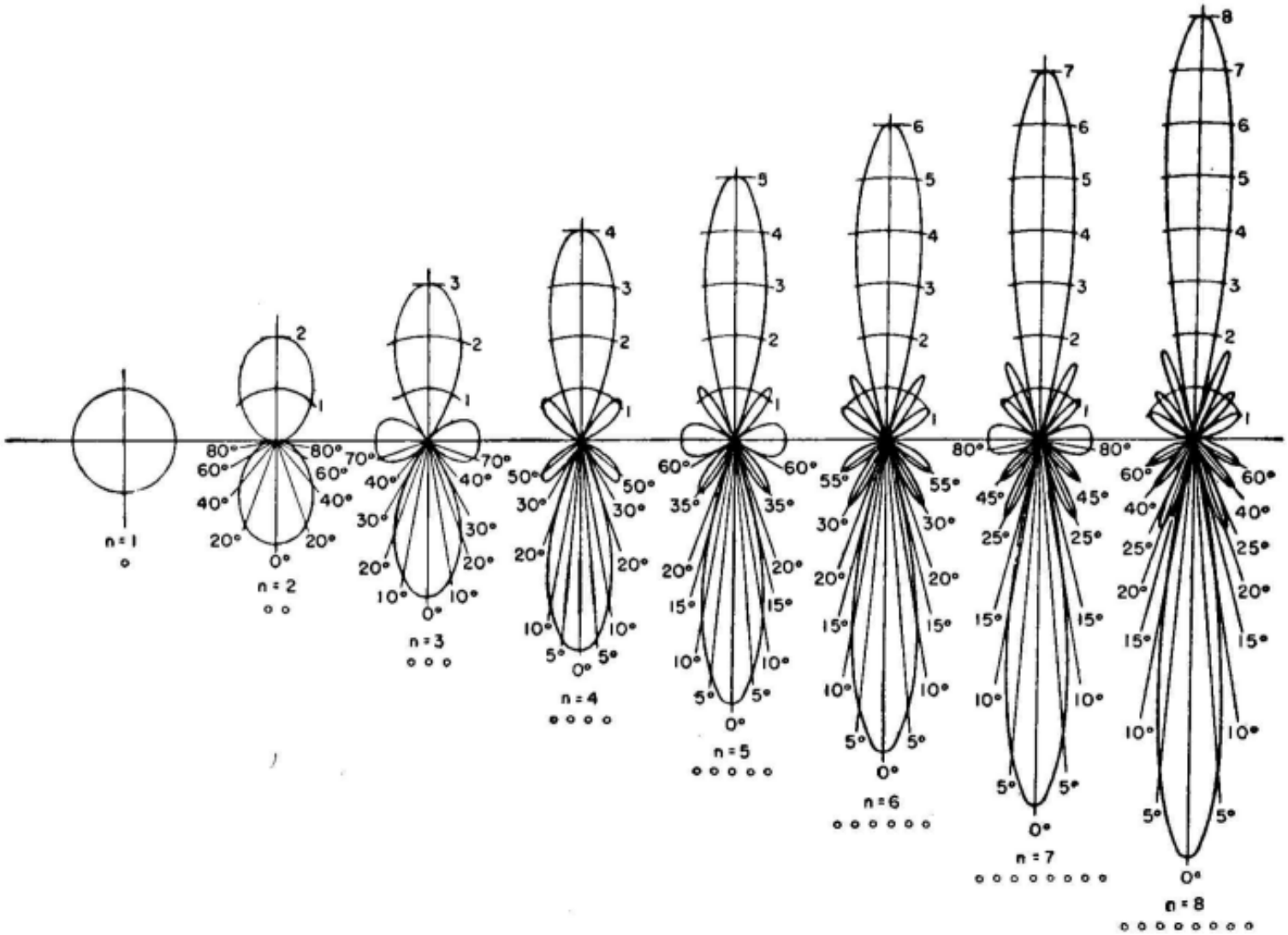




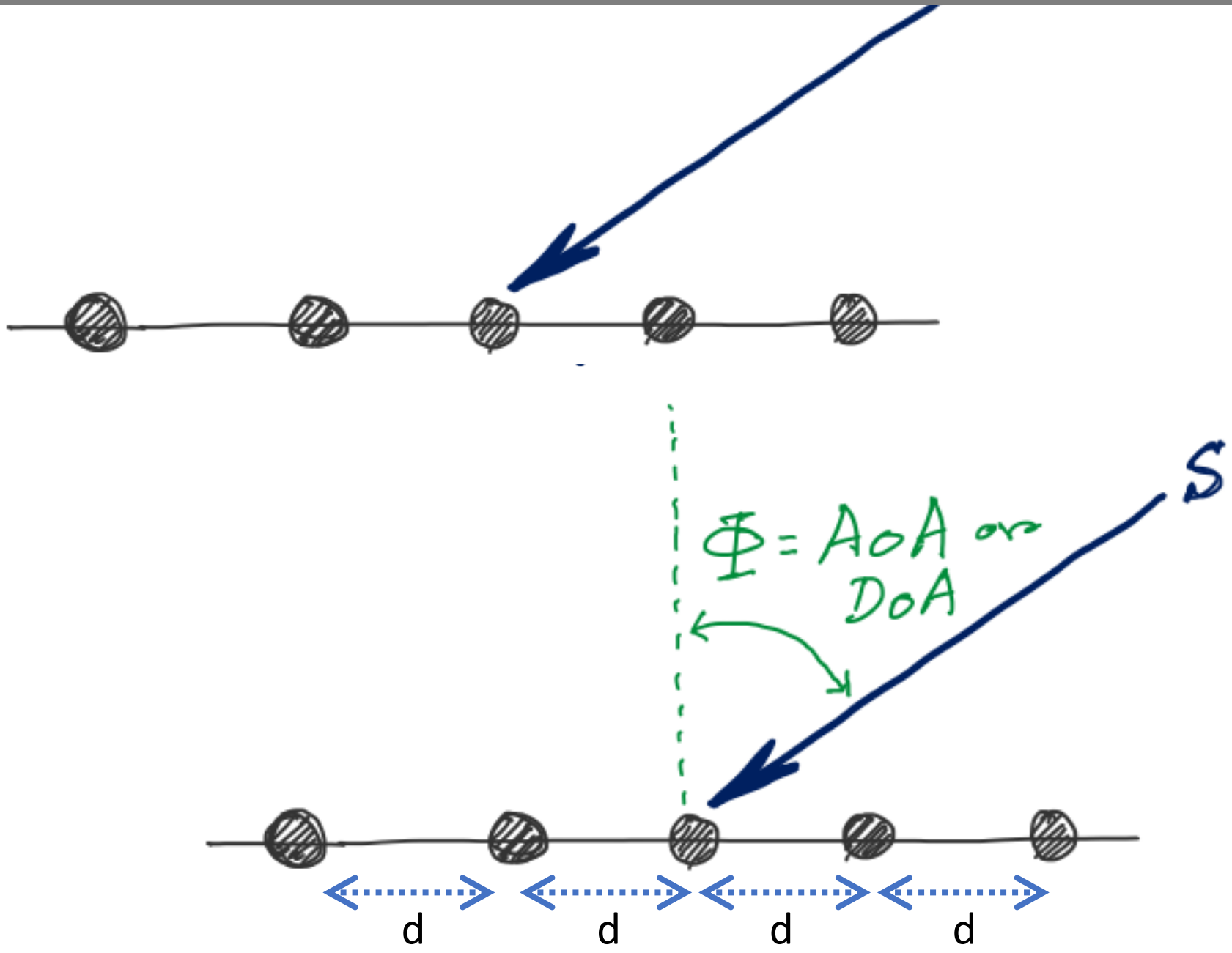
Figure 6: Pattern construction for $N = 5$, $d = \lambda/2$

Spatial Sensing with an Array

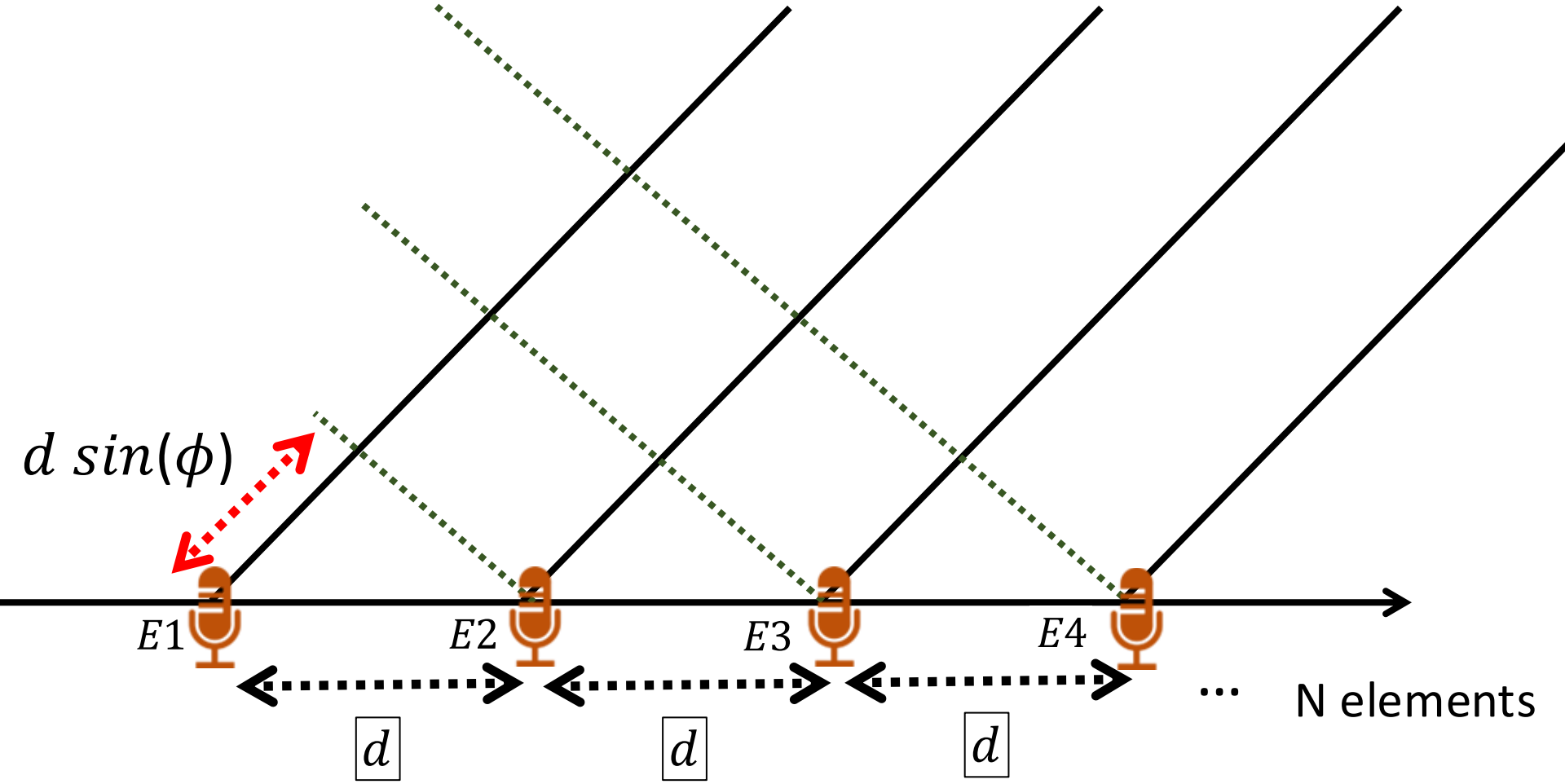


Uniform Linear Array (ULA)

Angle of Arrival (AoA) or Direction of Arrival (DoA)

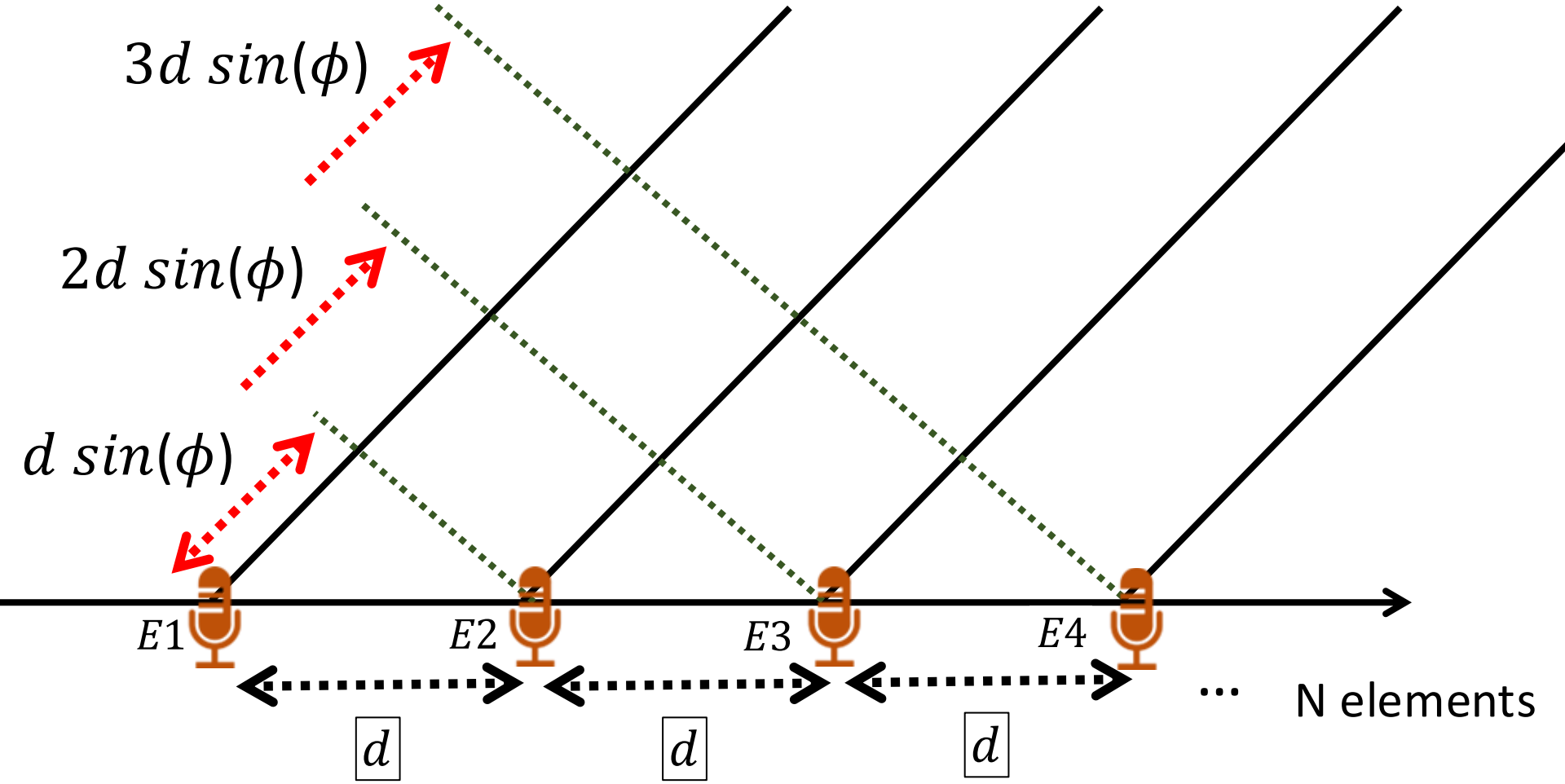


DoA in Microphone array



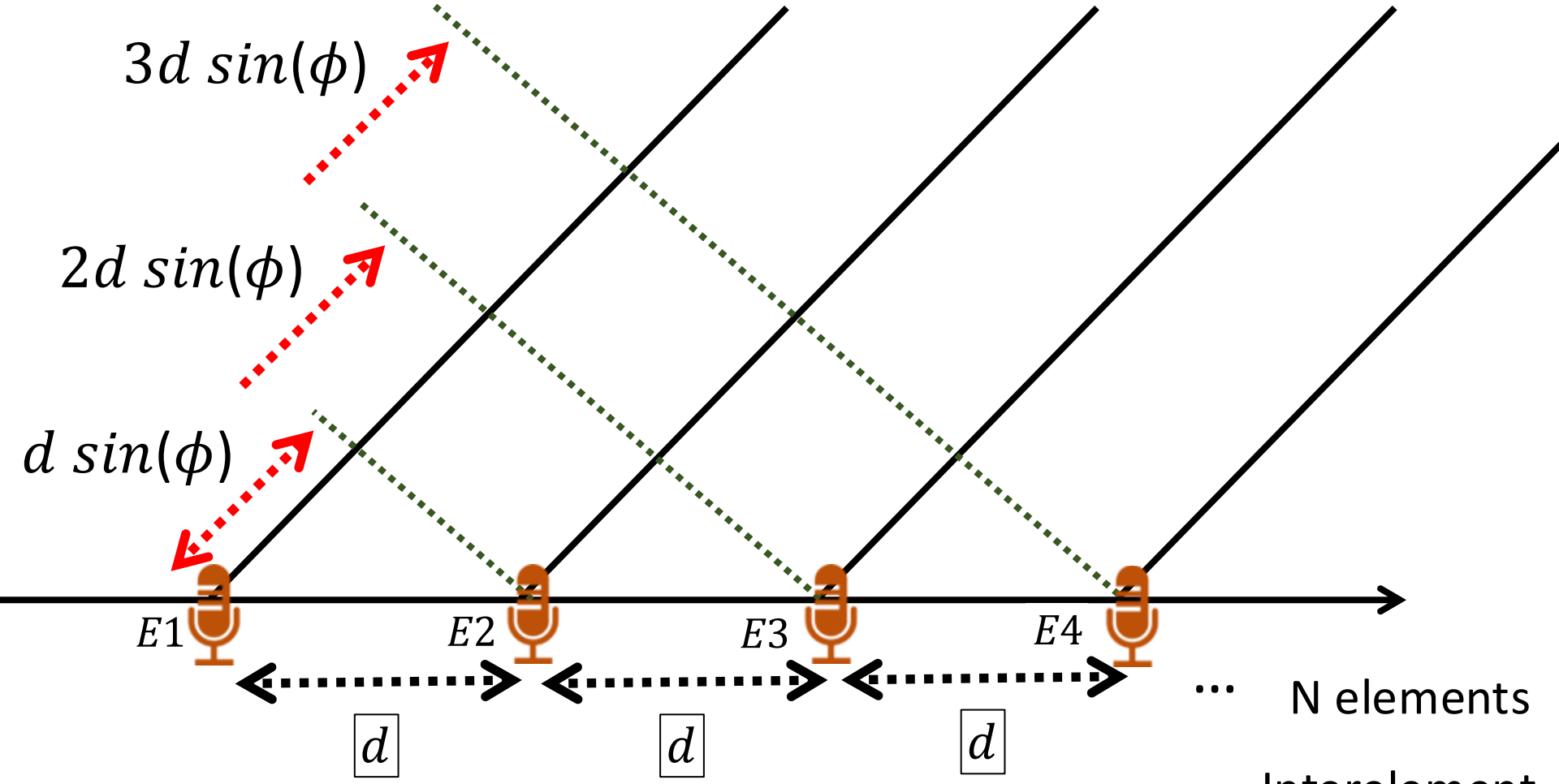
Interelement
distance = d

DoA in Microphone arrays



Interelement
distance = d

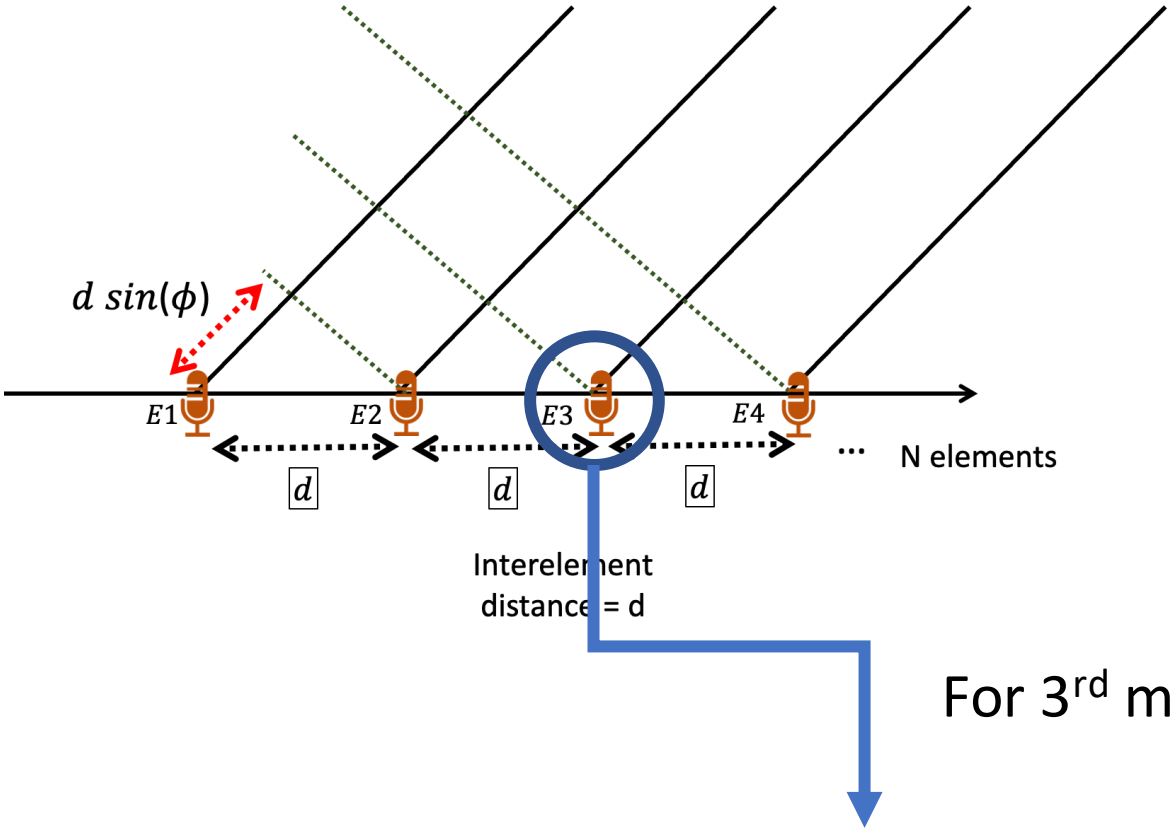
DoA in Microphone arrays



Path difference with nth element: $(n-1)d\ sin(\phi)$

Phase difference with nth element: $(n-1)d\ sin(\phi)\ 2\Pi/\lambda$

DoA in Microphone arrays



For 3rd mic, n == 3

Receive signal at nth element/microphone:

$$Y_n = S\, e^{j(n-1)d\ \sin(\phi)\frac{2\pi}{\lambda}}$$

Real samples to Complex samples

A common way to convert a real signal x(t) into a complex signal z(t) is to construct its "analytic signal" using **Hilbert transform**:

$$z(t) = x(t) + j\,\mathcal{H}\{x(t)\}$$

Here, is the **Hilbert transform** of x(t). It produces a quadrature component: each positive-frequency component is shifted by 90 degree.

Example:

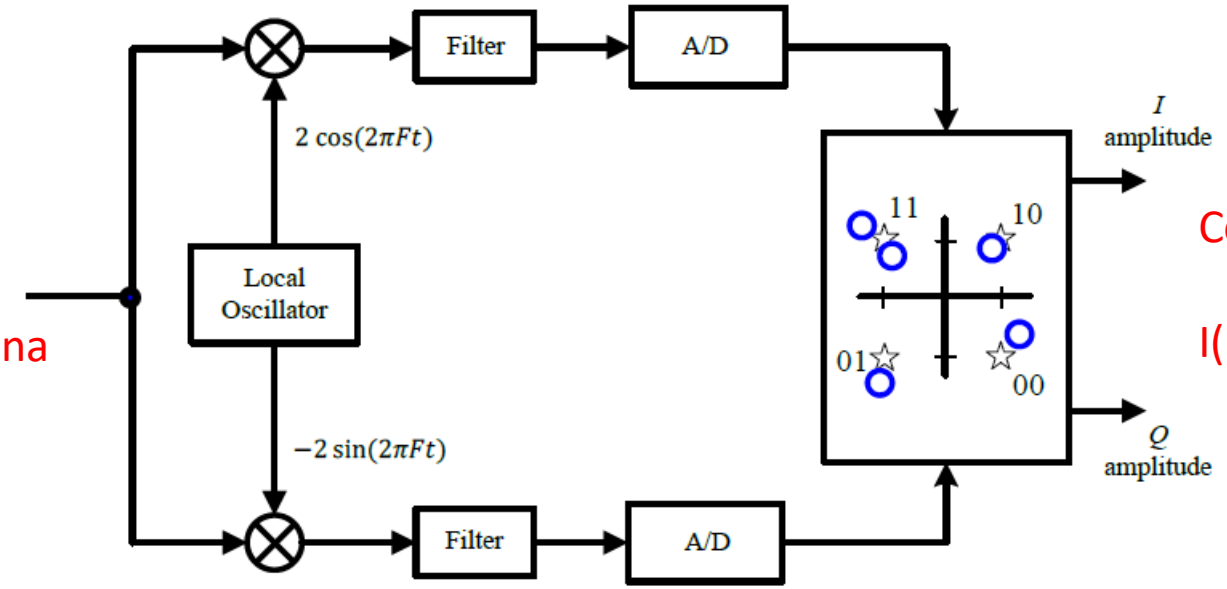
$$x(t) = \cos(2\pi f_0 t) \quad \Longrightarrow \quad \mathcal{H}\{x(t)\} = \sin(2\pi f_0 t),$$

$$z(t) = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t) = e^{j2\pi f_0 t}.$$

Real samples to Complex samples

I-Q Sampling

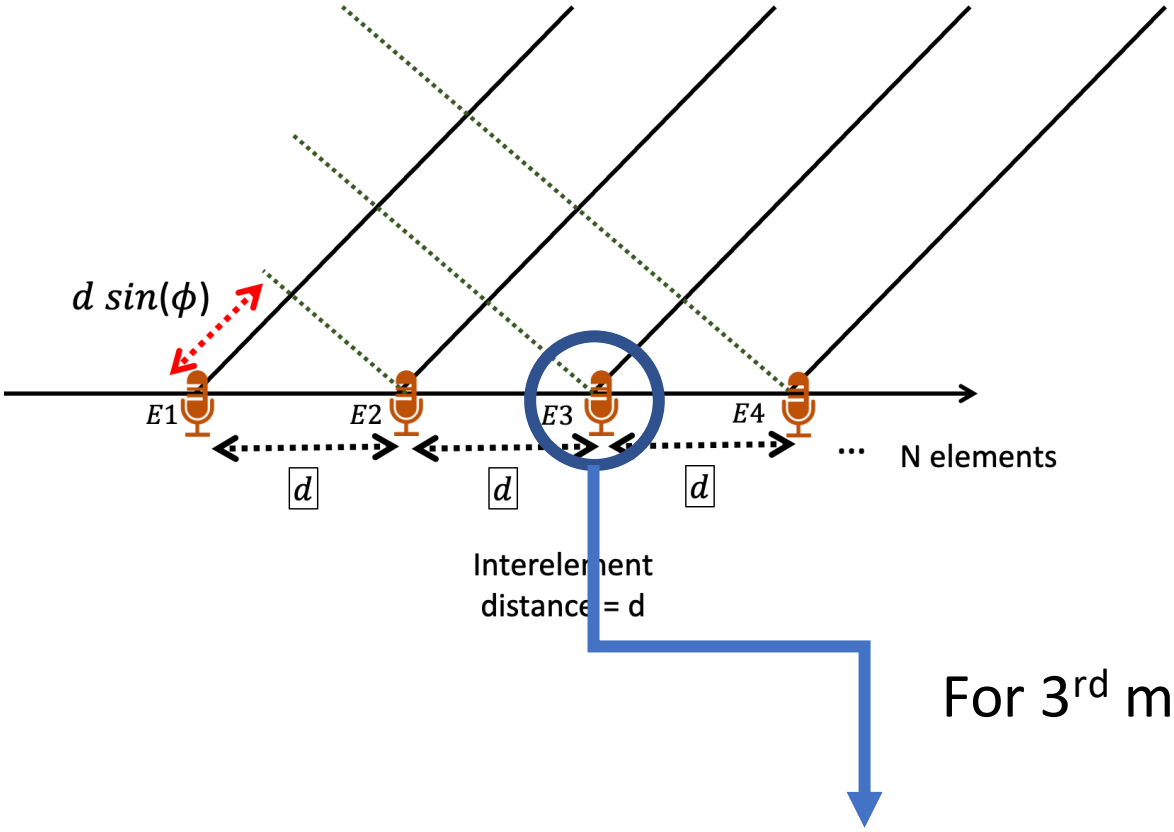
Analog real signal from sensor/antenna



Complex sequence is:

I(n) + Q(n)

DoA Estimation: Delay-sum



For 3rd mic, n == 3

Receive signal at nth element/microphone:

$$Y_n = S\, e^{j(n-1)d\ \sin(\phi)\frac{2\pi}{\lambda}}$$

From Array Samples to Steering Vectors

Incoming Signal = S, from an angle of ϕ

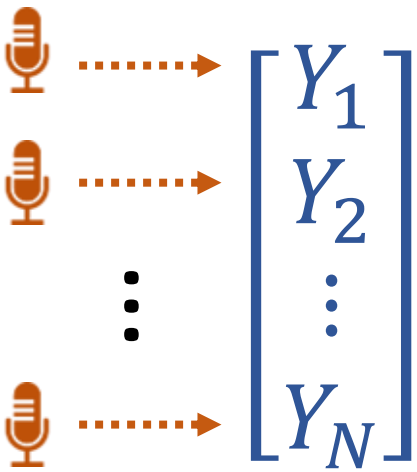
"S" is a complex scalar representing the source signal's amplitude and phase at the reference array element at a particular time when the "spatial sample" is taken.

Receive signal at nth element/microphone:

$$Y_n = S\, e^{j(n-1)d\,\sin(\phi)\frac{2\pi}{\lambda}}$$

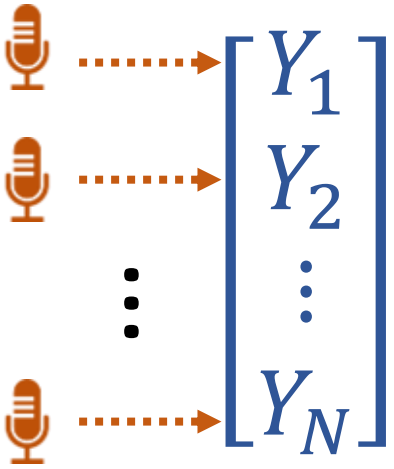
From Array Samples to Steering Vectors

Received signals from all elements of the array:



From Array Samples to Steering Vectors

Received signals from all elements of the array:



$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\boxed{Y = S\,\mathbf{a}(\phi)}$$

Steering vector

From Array Samples to Steering Vectors



$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\boxed{Y = S\,\mathbf{a}(\phi)}$$

$$\alpha = e^{j\,d\sin(\phi)\frac{2\pi}{\lambda}}$$

$$\mathbf{a}(\phi) = \begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \alpha^2 \\ \vdots \\ \alpha^{N-1} \end{bmatrix}$$

Multiple DoAs and Steering Matrix

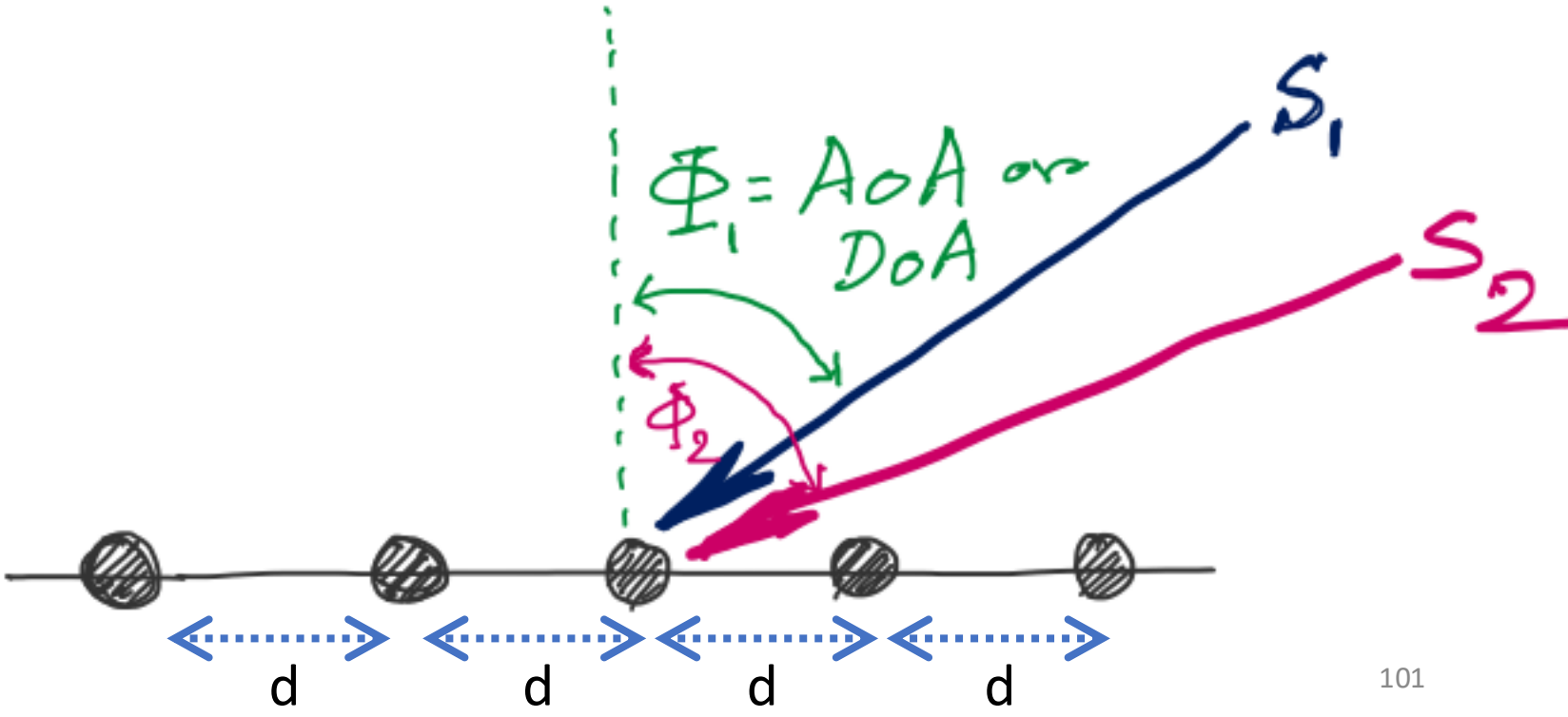
Array specification:

Number of elements = N

Inter-element distance = d

Frequency of interest = F_c

wavelength = λ



Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix}$$

$$= S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\mathbf{a}(\phi_i) = \begin{bmatrix} \alpha_i^0 \\ \alpha_i^1 \\ \alpha_i^2 \\ \vdots \\ \alpha_i^{N-1} \end{bmatrix}, \qquad \alpha_i = e^{j\,d\sin(\phi_i)\frac{2\pi}{\lambda}}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} \alpha_1^0 & \alpha_2^0 \\ \alpha_1^1 & \alpha_2^1 \\ \alpha_1^2 & \alpha_2^2 \\ \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \mathbf{AS}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} \alpha_1^0 & \alpha_2^0 \\ \alpha_1^1 & \alpha_2^1 \\ \alpha_1^2 & \alpha_2^2 \\ \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \mathbf{AS}$$

For a N-element array and M sources

$$Y_{N\times 1} = \mathbf{A}(\boldsymbol{\phi})_{N\times M}\, \mathbf{S}_{M\times 1} + \mathbf{W}_{N\times 1}$$

Steering matrix

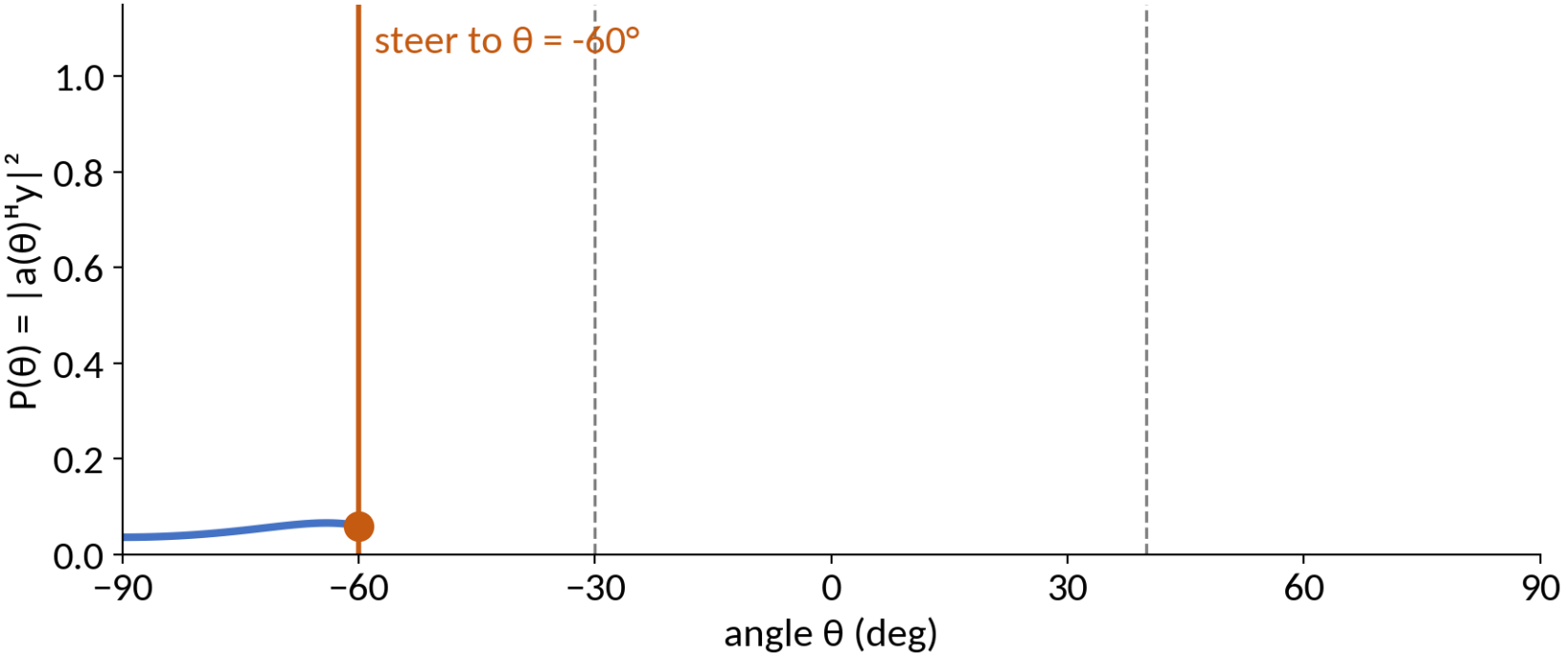
Noise

Solve the standard array problem first

Could we simply try every direction?

Try every direction: the beam scan

$$P(\theta) = | a(\theta)^H y |^2$$

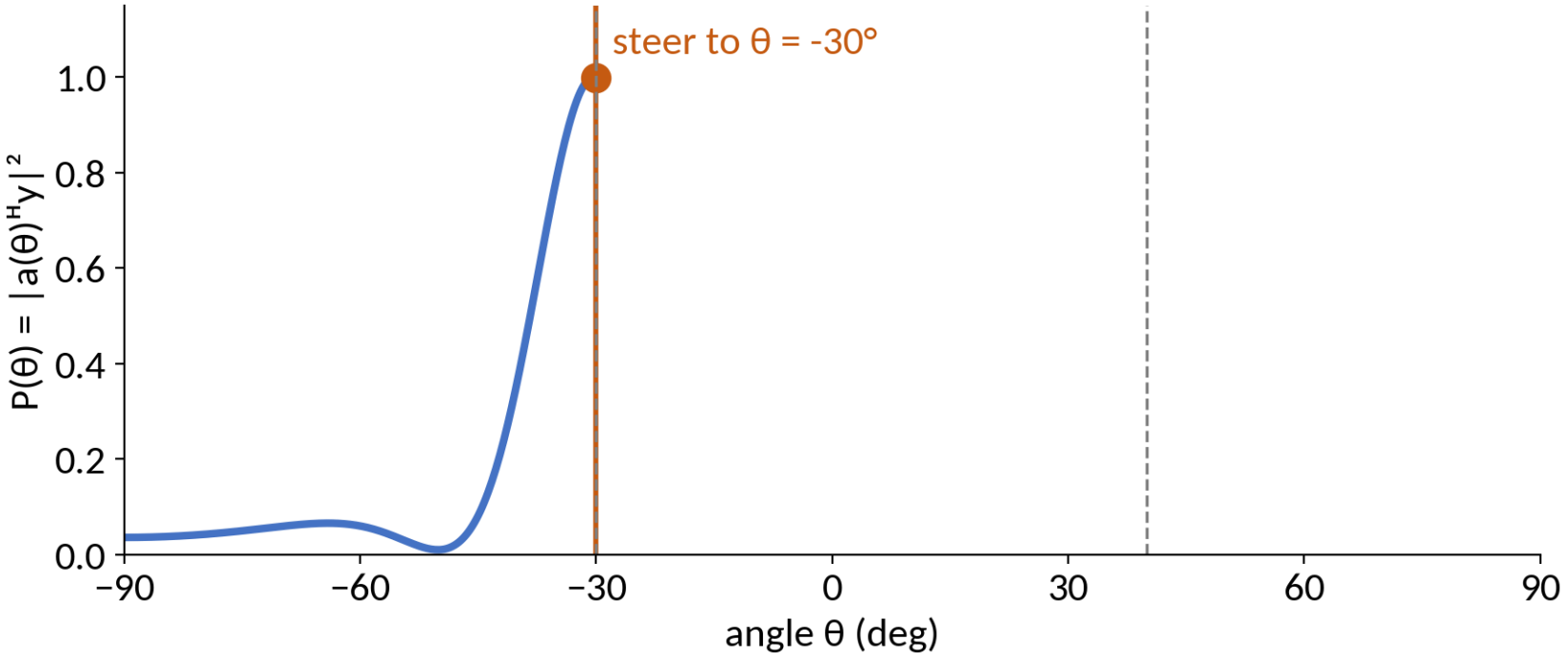


Start at −90° and steer the array one direction at a time.

Before we scan: what should P(θ) look like for our two speakers?

Try every direction: the beam scan

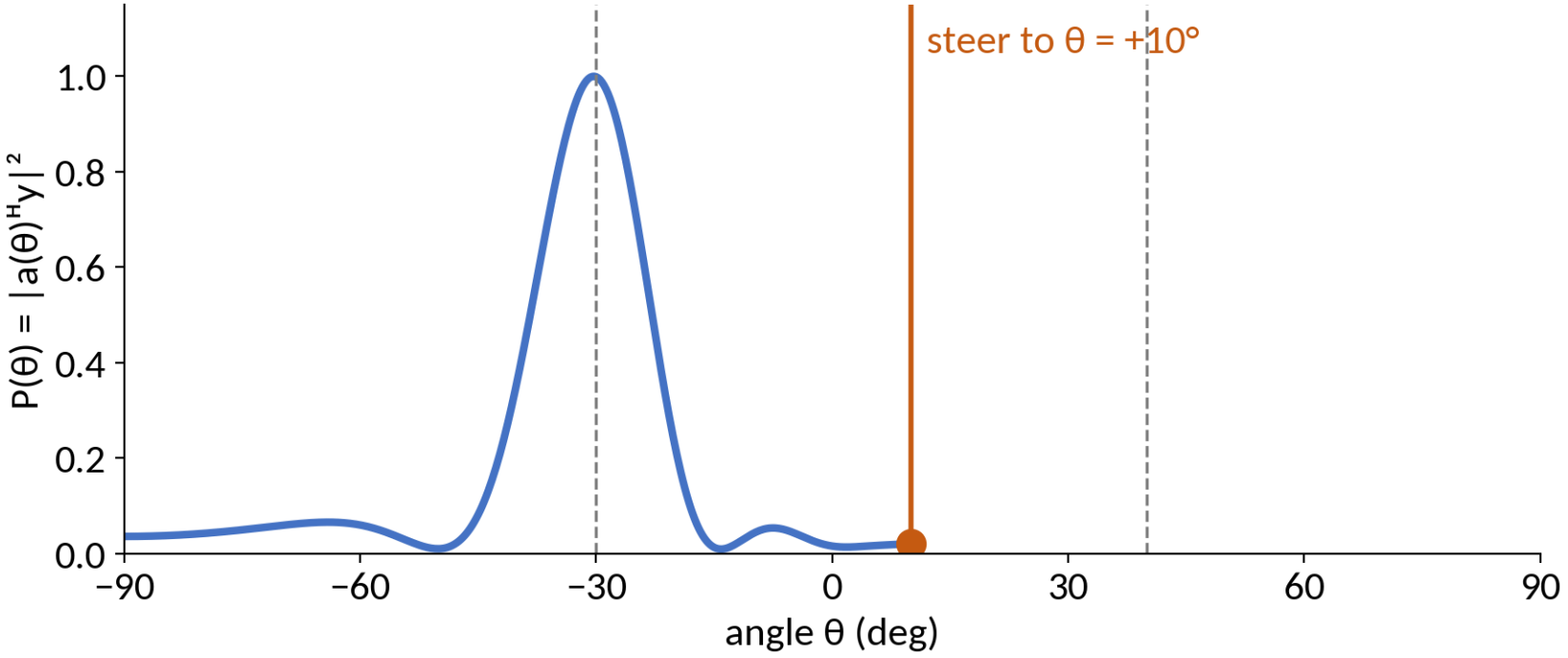
$$P(\theta) = |\, a(\theta)^H\, y \,|^2$$



At −30° the candidate matches source 1: the output jumps.

Try every direction: the beam scan

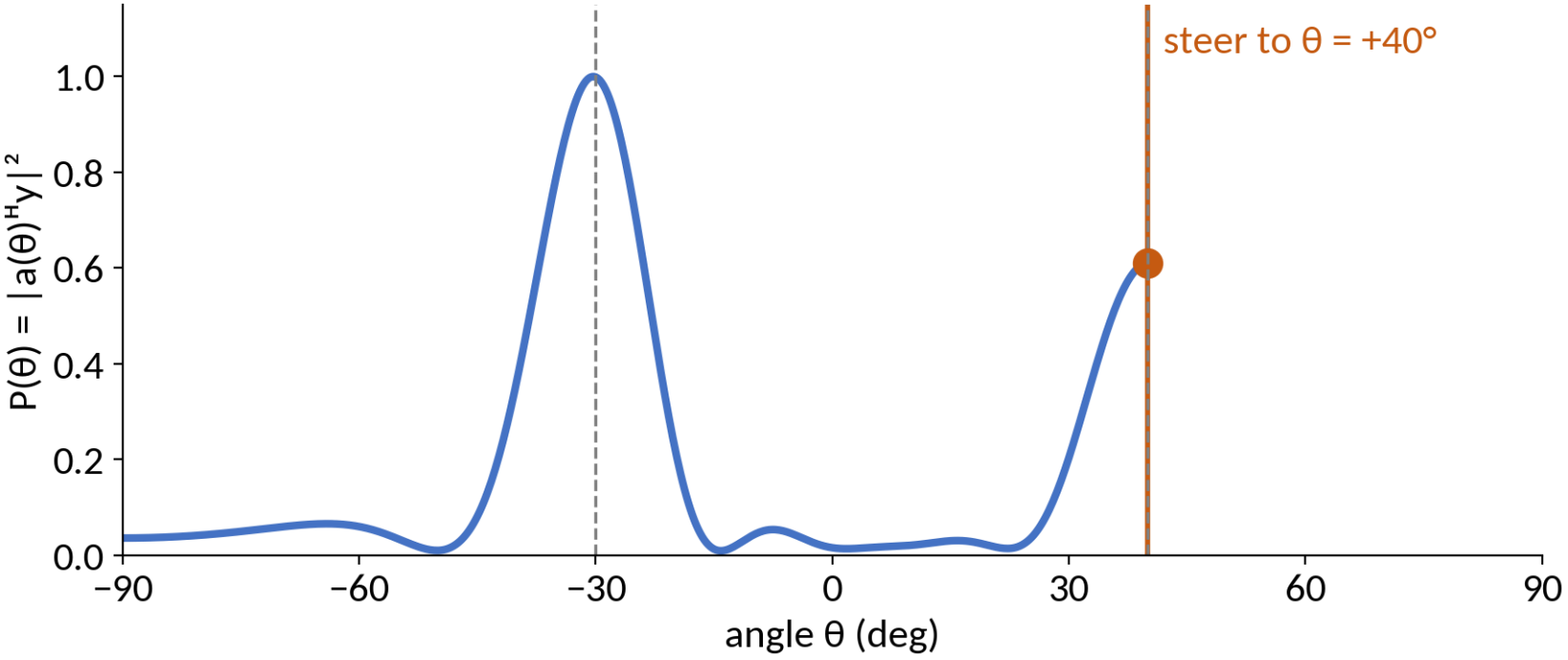
$$P(\theta) = |\, a(\theta)^H y \,|^2$$



Between the sources: small side lobes only.

Try every direction: the beam scan

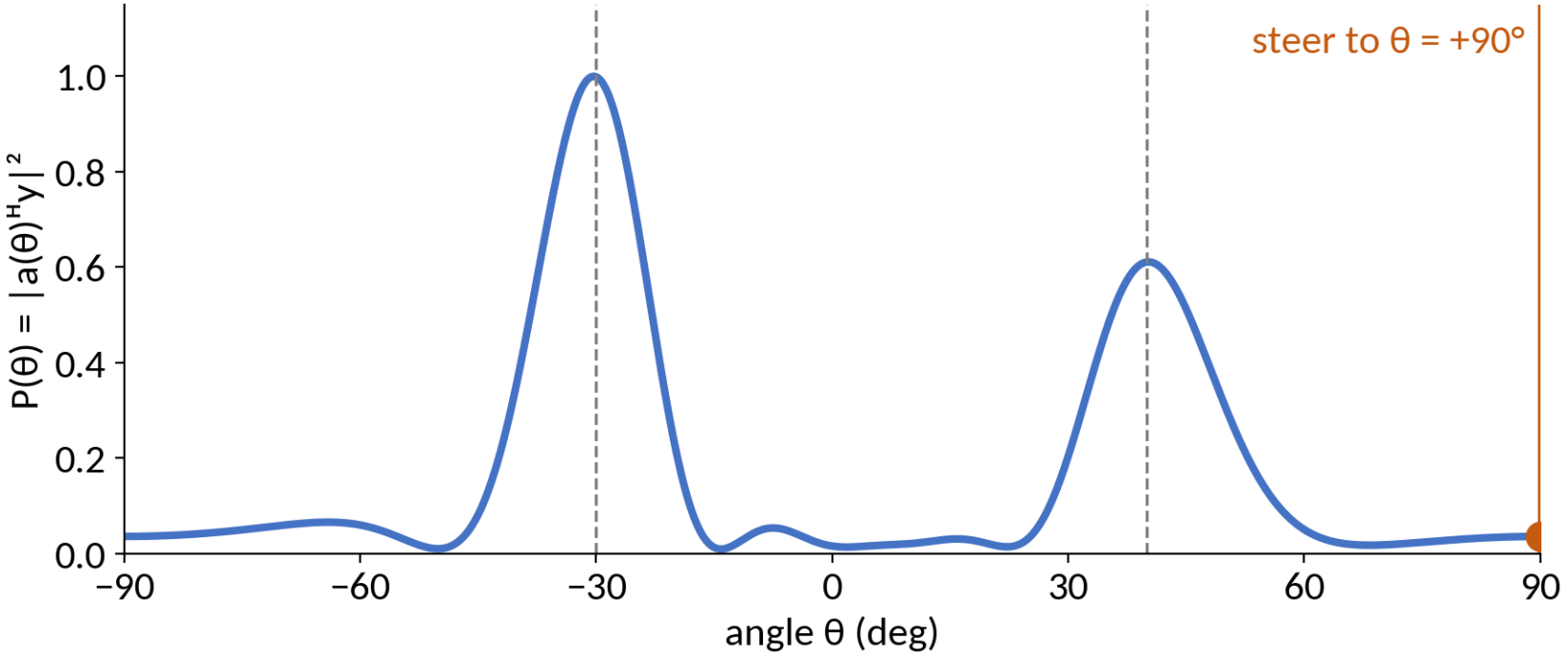
$$P(\theta) = |\, a(\theta)^H\, y \,|^2$$



At +40° a second match: source 2.

Try every direction: the beam scan

$$P(\theta) = | a(\theta)^H y |^2$$

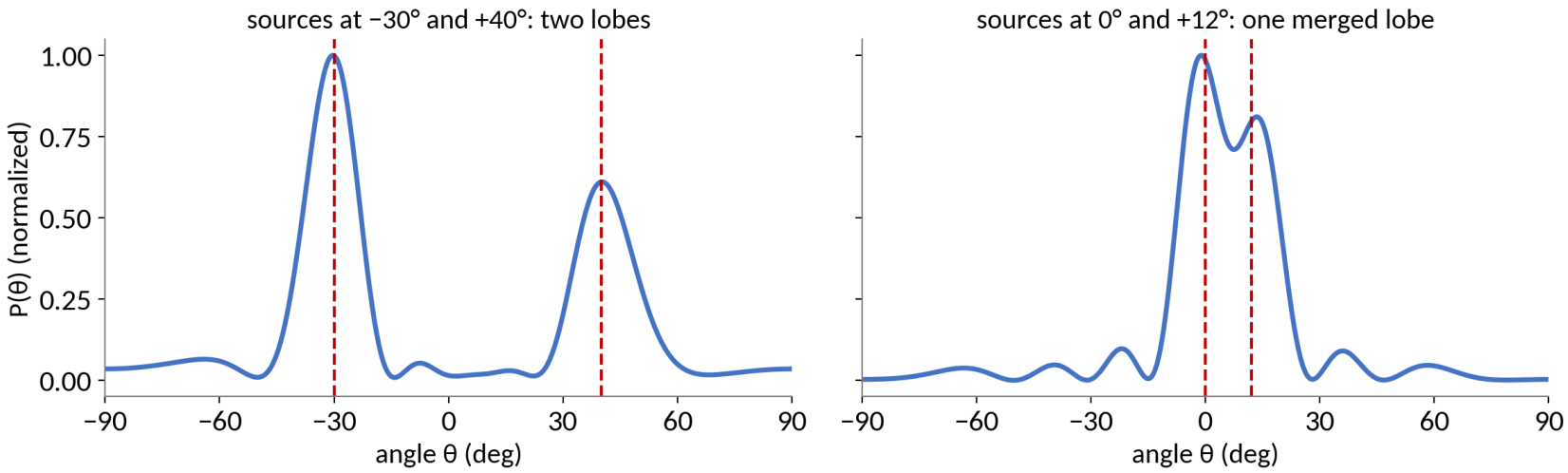


Full scan: two lobes, one per source.

Classical DoA matches measurements against predicted signatures.

Nearby sources expose a limitation

$$y = \Sigma_i \, a(\theta_i) \, s_i + n$$

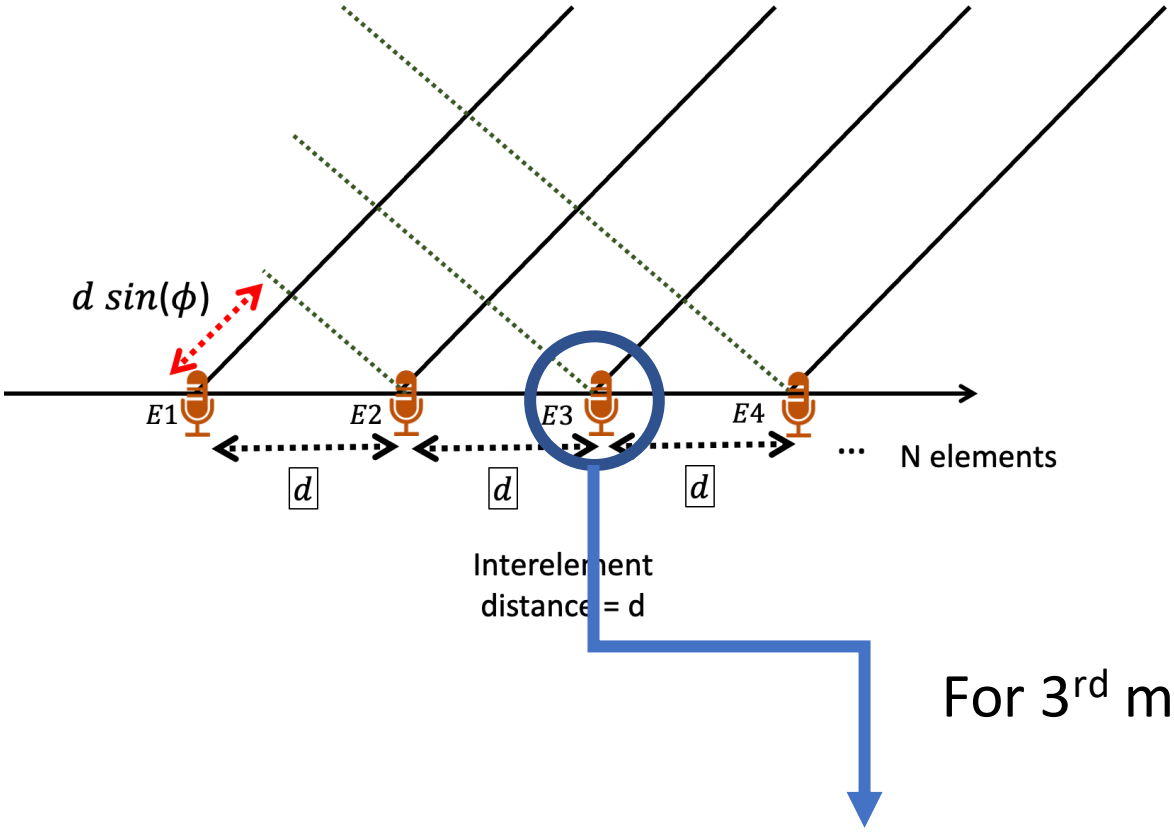


What else do we know about the scene that plain scanning is not using?

Only a small number of directions usually contain active sources.

DoA problem as learned optimization:
Recap what we learned before Fall break

DoA Estimation: Delay-sum



For 3rd mic, n == 3

Receive signal at nth element/microphone:

$$Y_n = S\, e^{j(n-1)d\ \sin(\phi)\frac{2\pi}{\lambda}}$$

Multiple DoAs and Steering Matrix

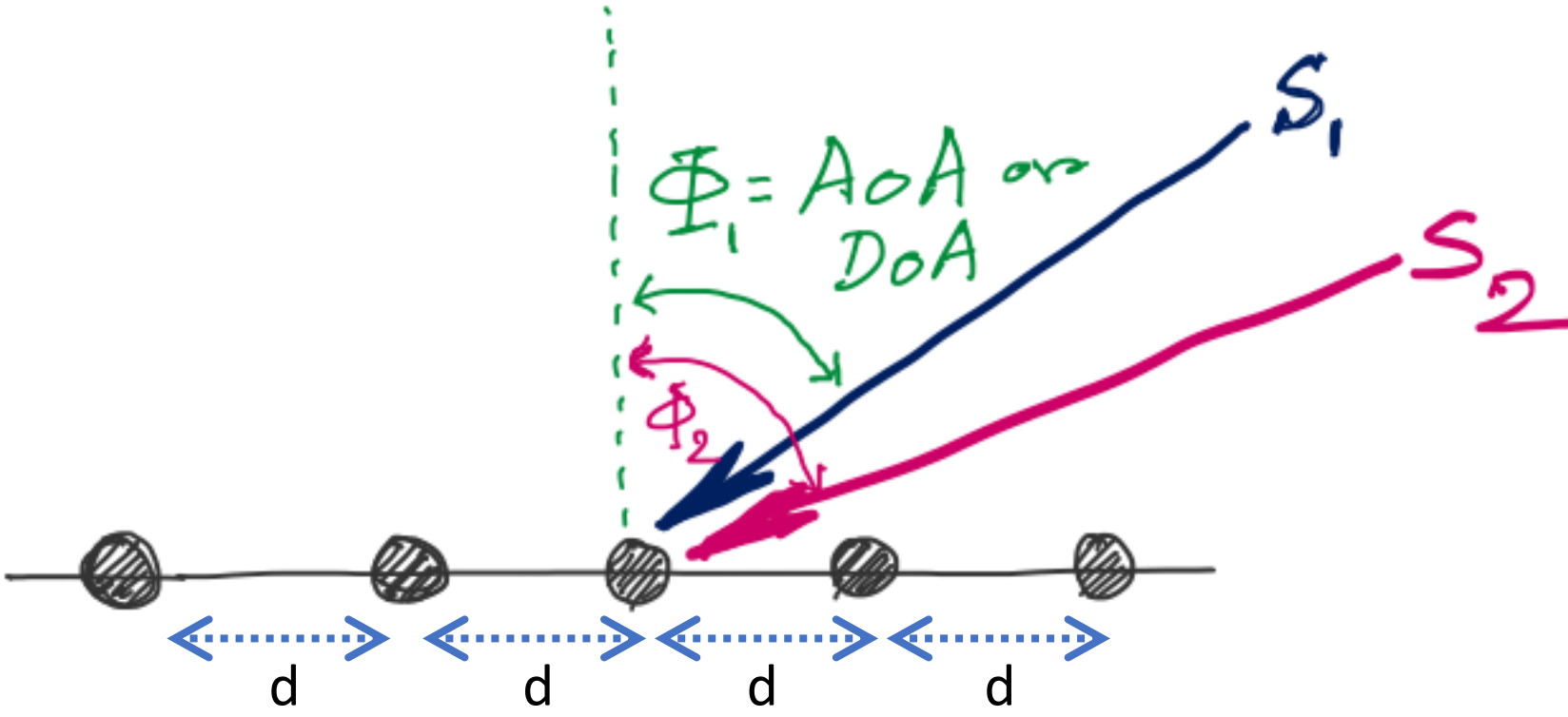
Array specification:

Number of elements = N

Inter-element distance = d

Frequency of interest = F_c

wavelength = λ



Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix}$$

$$= S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\mathbf{a}(\phi_i) = \begin{bmatrix} \alpha_i^0 \\ \alpha_i^1 \\ \alpha_i^2 \\ \vdots \\ \alpha_i^{N-1} \end{bmatrix}, \qquad \alpha_i = e^{j\,d\sin(\phi_i)\frac{2\pi}{\lambda}}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = S_1 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_1)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_1)\frac{2\pi}{\lambda}} \end{bmatrix} + S_2 \begin{bmatrix} e^{j0} \\ e^{jd\sin(\phi_2)\frac{2\pi}{\lambda}} \\ \vdots \\ e^{j(N-1)d\sin(\phi_2)\frac{2\pi}{\lambda}} \end{bmatrix}$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} \alpha_1^0 & \alpha_2^0 \\ \alpha_1^1 & \alpha_2^1 \\ \alpha_1^2 & \alpha_2^2 \\ \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \mathbf{AS}$$

Multiple DoAs and Steering Matrix

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} = \begin{bmatrix} \alpha_1^0 & \alpha_2^0 \\ \alpha_1^1 & \alpha_2^1 \\ \alpha_1^2 & \alpha_2^2 \\ \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \mathbf{AS}$$

For a N-element array and M sources

$$Y_{N\times 1} = \mathbf{A}(\boldsymbol{\phi})_{N\times M}\, \mathbf{S}_{M\times 1} + \mathbf{W}_{N\times 1}$$

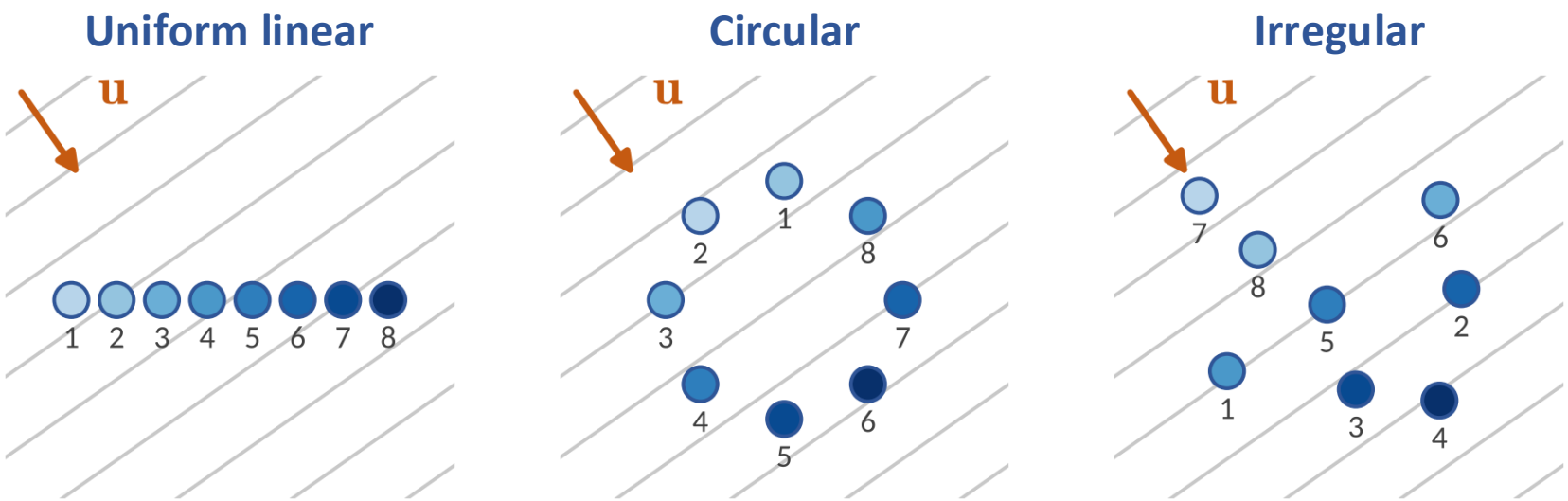
Steering matrix

Noise

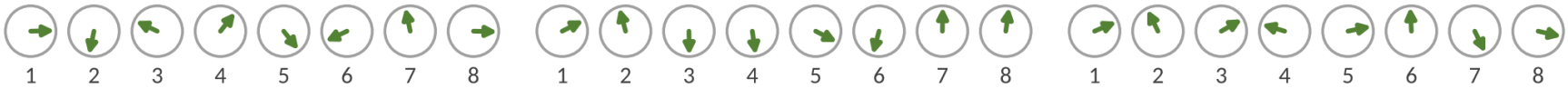
Steering Vector for Arbitrary Array Geometry

Does a Steering Vector Work Only for Linear Arrays?

Same plane wave (direction u) hits three 8-mic arrays



shading = arrival order (light = earlier, dark = later)

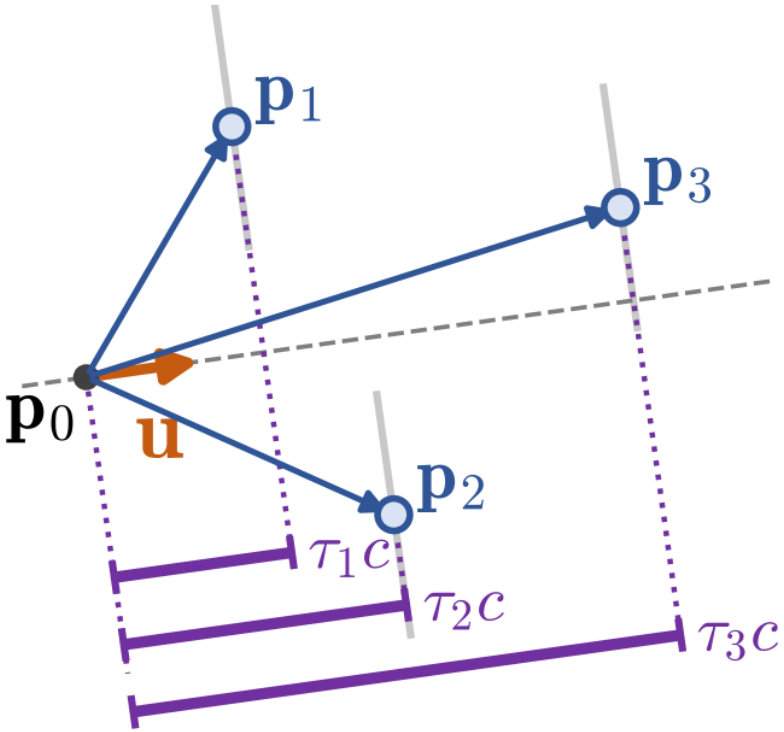


phase of each entry $a_m(u)$ — same wave, three different phase patterns

$$a(u) = [\, a_1(u),\ a_2(u),\ \ldots,\ a_M(u)\,]^T$$

A steering vector = how an array responds to a wave from direction u — any geometry.

Array Geometry Determines the Propagation Delay



Mic positions: $p_m = [x_m, y_m, z_m]^T$
Reference point: p_0
u = unit propagation direction

projected distance $u^T(p_m - p_0)$

$\div c$

delay $\tau_m = u^T(p_m - p_0) / c$

$\times 2\pi f$ (phase)

phase $a_m(u) = \exp(-j2\pi f \tau_m)$

Mics farther along u are reached later → larger delay → more phase rotation

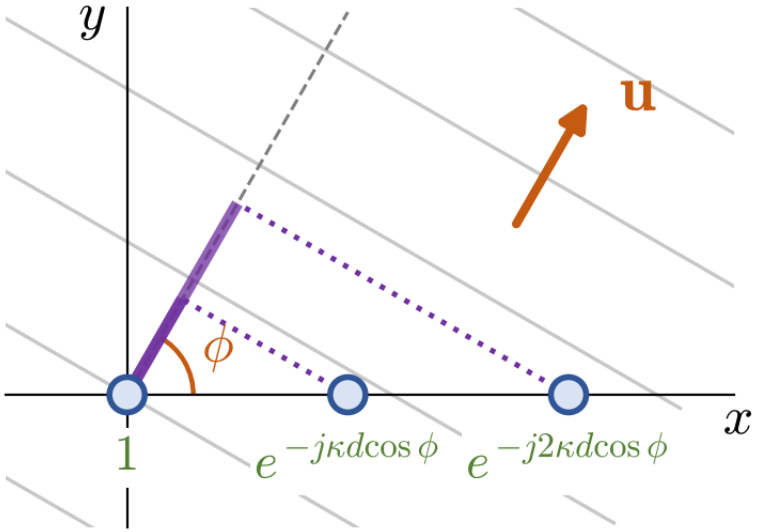
$$a_m(u) = \exp(-j (2\pi f / c) u^T(p_m - p_0))$$

Physics gives the steering vector; geometry enters only via the mic positions p_m.

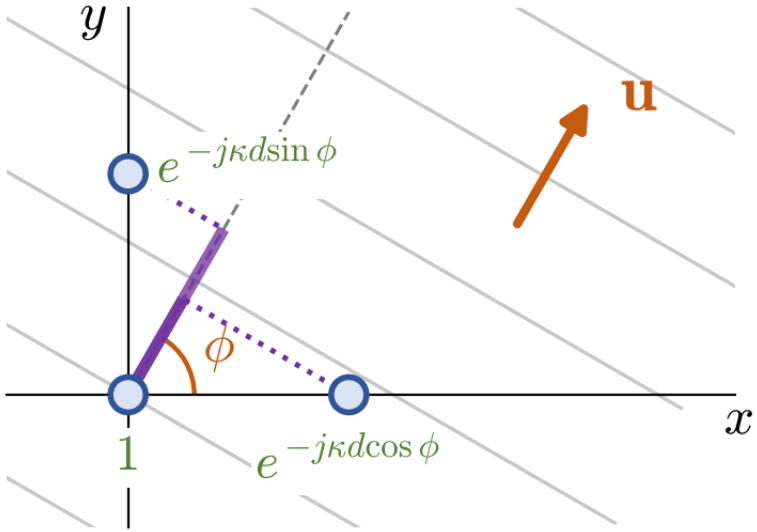
Different Geometries, Different Steering Vectors

Convention: u = (cos ϕ, sin ϕ), ϕ measured from +x axis, κ = 2πf / c

ULA: p = (0,0), (d,0), (2d,0)



L-shape: p = (0,0), (d,0), (0,d)



	ULA	L-shape
mic 1 (reference)	1	1
mic 2 at (d, 0)	$e^{-j\kappa d\cos\varphi}$	$e^{-j\kappa d\cos\varphi}$
mic 3	$e^{-j2\kappa d\cos\varphi}$	$e^{-j\kappa d\sin\varphi}$
depends on	cos ϕ only → ϕ and −ϕ look identical	cos ϕ and sin ϕ → full azimuth

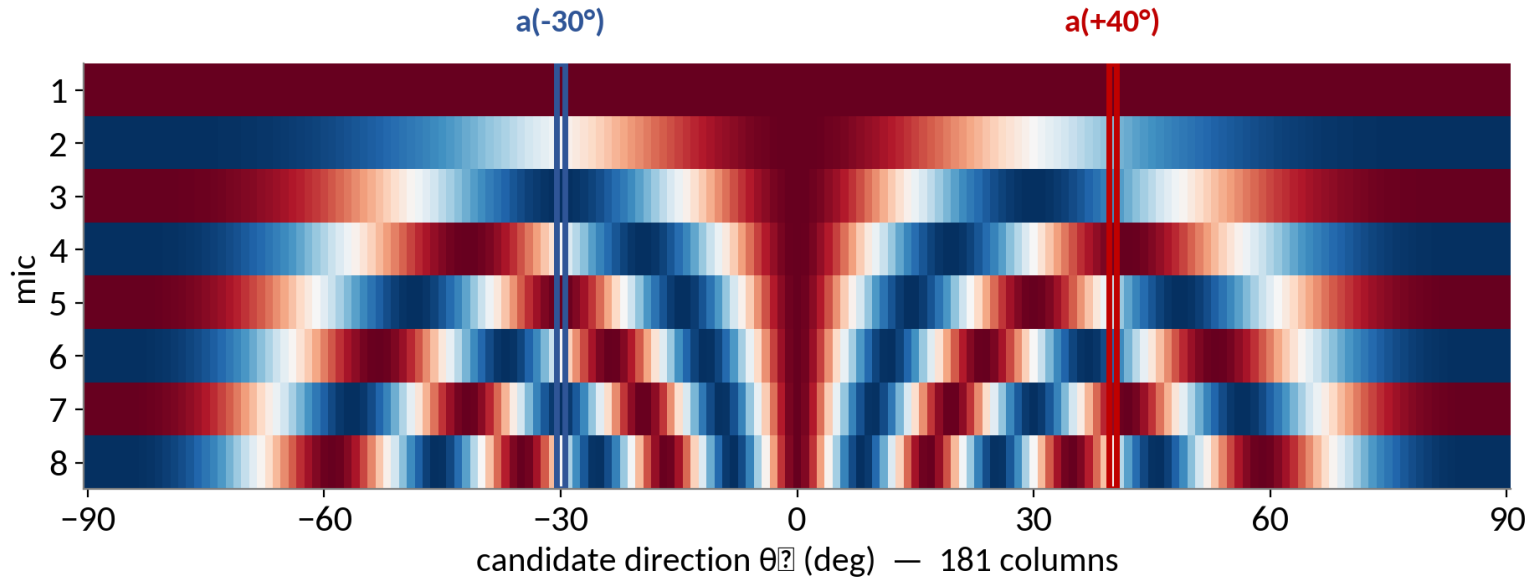
If the mic geometry changes, do we need a whole new inverse-problem algorithm?

No — only the forward model changes: any geometry's steering vectors build a matrix A.

DoA as a sparse inverse problem

Discretize the angular world

$$A = [\, a(-90^\circ),\ a(-89^\circ),\ \ldots,\ a(+90^\circ)\,]$$



8 microphones × 181 candidate angles: A is 8 × 181. Column k is the steering vector for θ_k.

Array processing becomes the familiar inverse problem y = Ax + n.

Pieces of y = Ax + n

$$y = A x + n$$

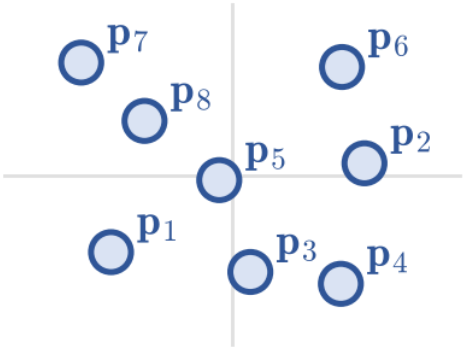
What is the physical meaning and size of y, A, x, n?

Symbol	Size	Physical meaning
y	8 × 1	what the 8 microphones recorded (one snapshot)
A	8 × 181	predicted signature for every candidate angle
x	181 × 1	source amplitude at each candidate angle
n	8 × 1	sensor noise and unmodeled effects

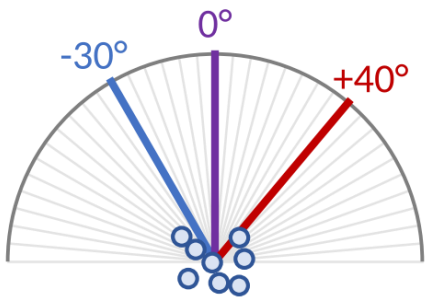
Same equation as the CT lectures; only the physics inside A changed.

From Any Array Geometry to the Same Inverse Problem

① **8 mics, any known geometry**

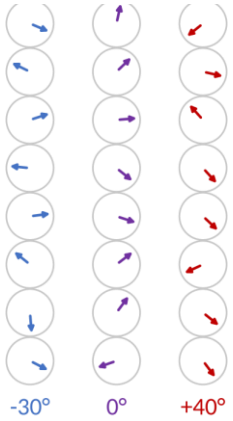


② **Azimuth grid −90° …90° (181)**



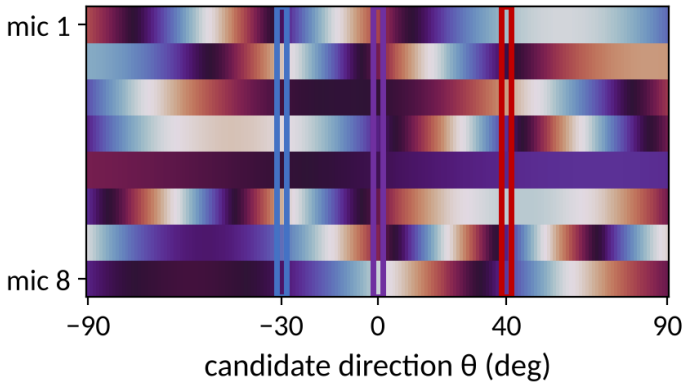
elevation known → 1-D grid, 1° steps

③ **a(θ) for each**



④ **Side by side → A = [a(−90°) … a(90°)]**

column = one candidate direction · row = one mic



⑤ **The same linear inverse problem**

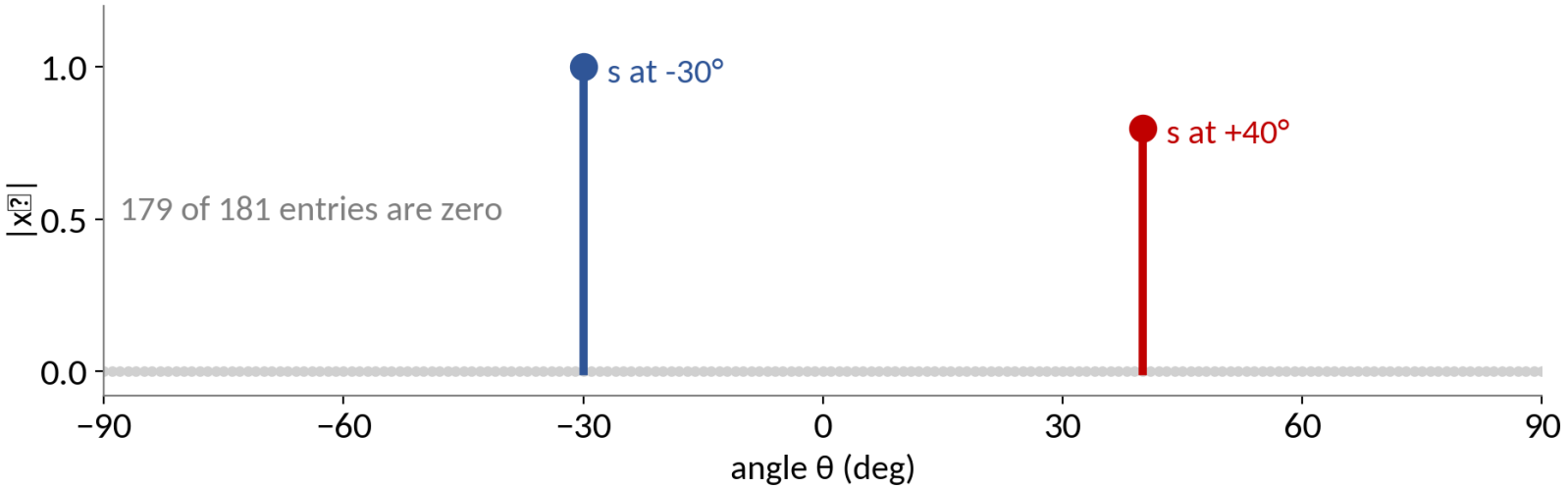
$$\underset{\mathbb{C}^8}{y} = \underset{\mathbb{C}^{8\times 181}}{A}\ \underset{\mathbb{C}^{181}}{x} + \underset{\mathbb{C}^8}{n}$$

8 mics	known model	181 amplitudes	noise

8 measurements, 181 possible source amplitudes. How can we reconstruct the actual source directions?

What does x look like physically?

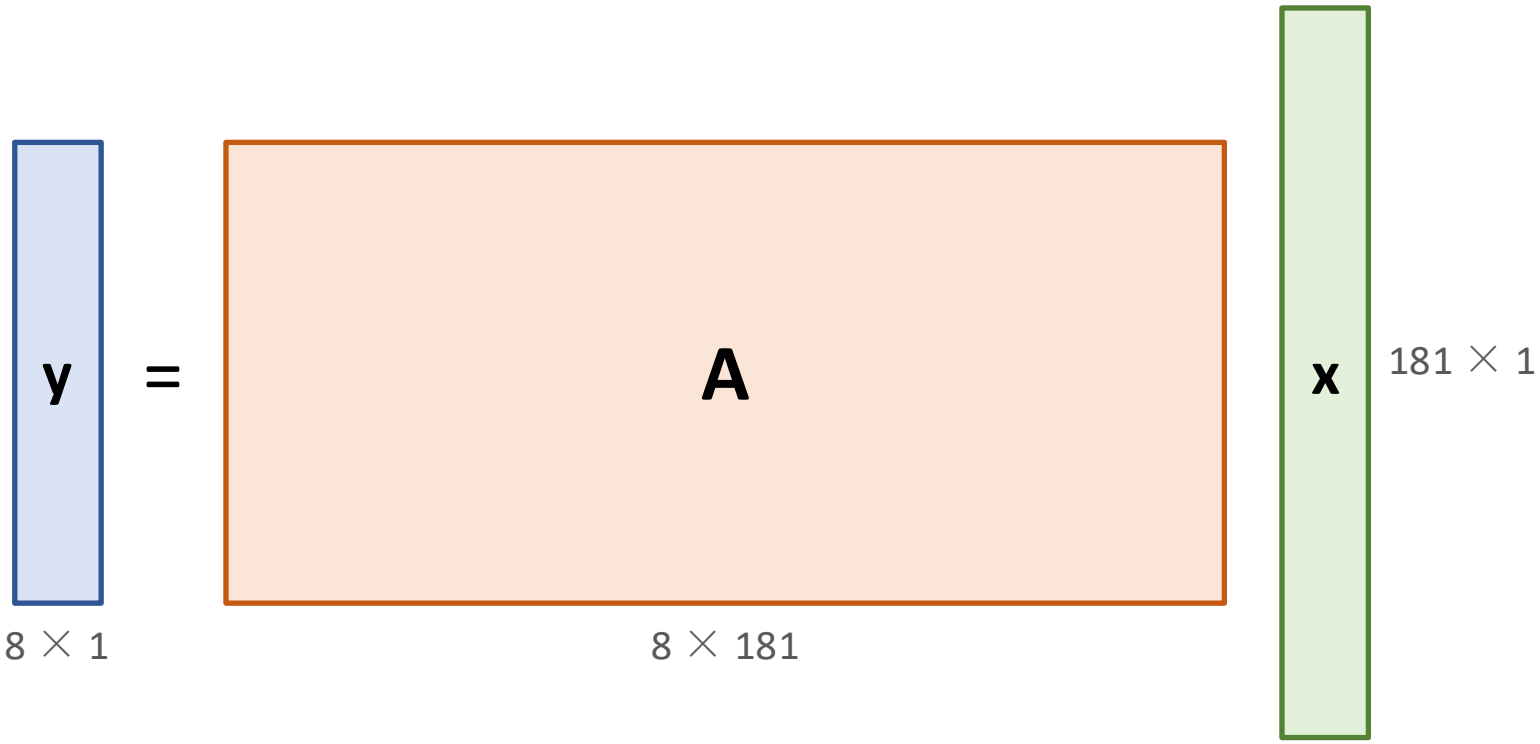
$$x = [\, 0, \ldots, s_1 \text{ at } -30^\circ, \ldots, s_2 \text{ at } +40^\circ, \ldots, 0 \,]^T$$



Why is x sparse?
Physical Answer: because there are few sources, not because an algorithm prefers it.

Sparsity is prior knowledge about the physical scene.

Why can't we just invert A?



8 equations, 181 unknowns: infinitely many x reproduce y exactly.

We need both measurement consistency and prior knowledge.

In the inverse-problem formulation you have been using,

$$\boxed{\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}}$$

the terms **overdetermined, underdetermined, and determined** simply compare the **number of independent measurements/equations** with the **number of unknowns**.

Suppose

$$\mathbf{A} \in \mathbb{R}^{m \times n}, \qquad \mathbf{x} \in \mathbb{R}^{n}, \qquad \mathbf{y} \in \mathbb{R}^{m}.$$

Here, m = number of measurements/equations and n = number of unknown parameters.

System	Relationship	Meaning	Typical solution
Overdetermined	$m > n$	More equations than unknowns	Least squares
Square / determined	$m = n$	Same number of equations and unknowns	$A^{-1}y$, if A invertible
Underdetermined	$m < n$	Fewer equations than unknowns	Need constraints/priors
Ill-posed / rank-deficient	$\mathrm{rank}(A) < n$	Some directions of x cannot be observed	Regularization/prior

The physical prior becomes an objective

Two requirements for a good reconstruction

Two requirements for a good reconstruction

$$\hat{x} = \operatorname{argmin}_x \; \tfrac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_1$$

Measurement term

$\tfrac{1}{2}\|Ax - y\|_2^2$

Does this source map reproduce what the microphones heard?

Prior term

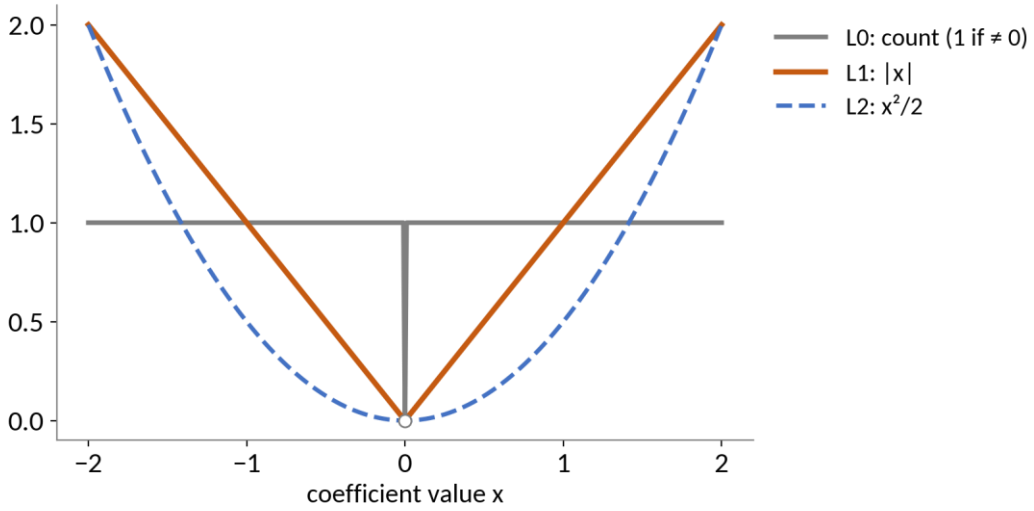
$\lambda\|x\|_1$

Does it contain only a few active directions?

λ sets the balance: trust the microphones versus trust the few-sources assumption.

Regularization is not an arbitrary penalty; it encodes scene knowledge.

Why L1?



What we want: count active directions

$$\|x\|_0 = \#\{\, k : x_k \neq 0 \,\}$$

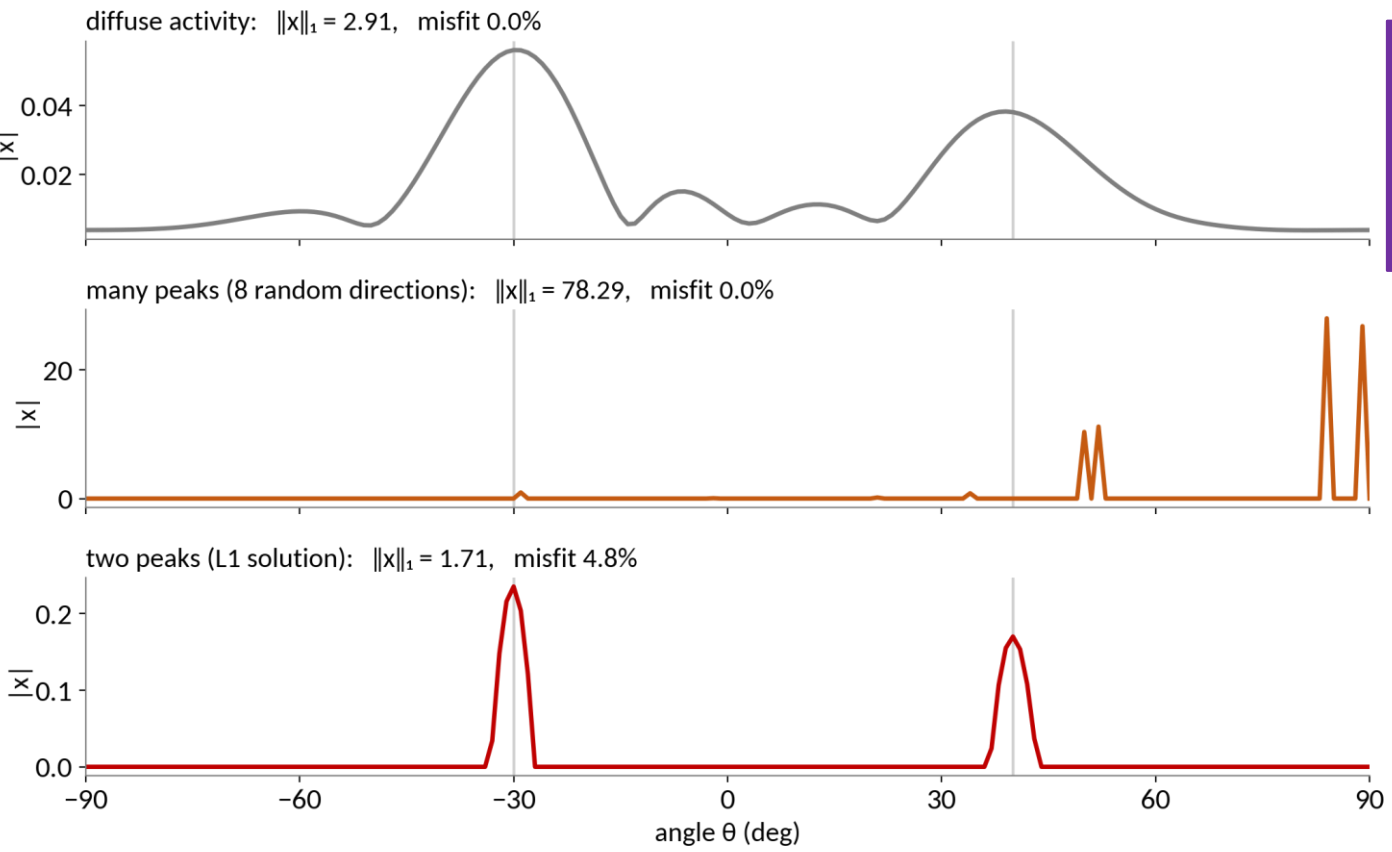
Combinatorial, hard to optimize. Tractable stand-in:

$$\|x\|_1 = \Sigma_i \, |x_i|$$

L1 keeps a sharp corner at zero like L0, and is convex. L2 is flat near zero, so it never pushes a coefficient all the way to 0.

L1 is the tractable stand-in for 'count the sources'.

Three source maps that all explain the mics



Which one matches a room with two speakers?

All three reproduce y to within the noise (misfit ≤ 5%). L1 picks the plausible one: smallest $\|x\|_1$.

L1 encourages the sparse angular spectrum we expect physically.

ISTA as a physical reconstruction algorithm

Listen to the microphones, then remember how few sources exist

How do we minimize the objective?

① Listen to the microphones

$$f(x) = \tfrac{1}{2}\|Ax - y\|_2^2$$

Smooth: its slope says how to change x to better explain the microphones.

② Remember: few sources

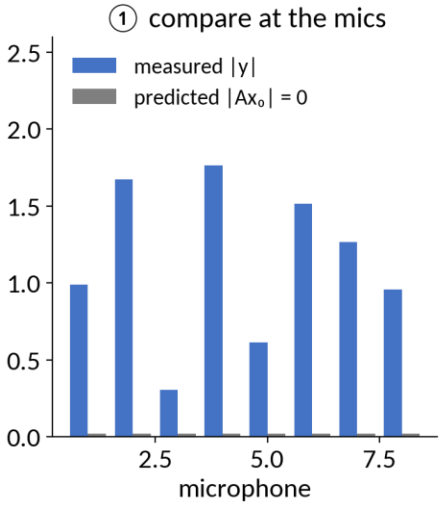
$$g(x) = \lambda\|x\|_1$$

Has a corner at 0: handled by a simple per-angle rule, not a slope.

We will alternate: listen to the measurements, then enforce what we know about the scene.

ISTA will be introduced as two understandable operations.

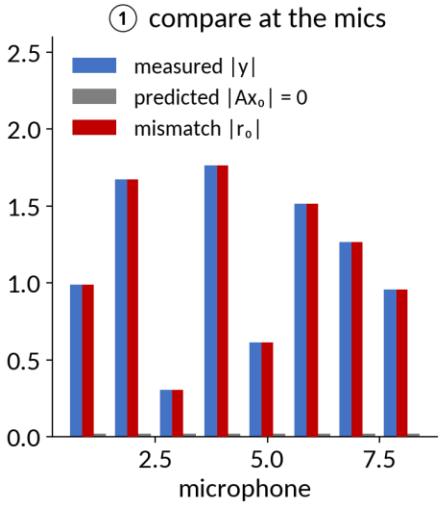
One ISTA iteration, step by step



$$\hat{y}_0 = A\, x_0$$

Start from $x_0 = 0$. Predict what the mics should hear: $Ax_0 = 0$.

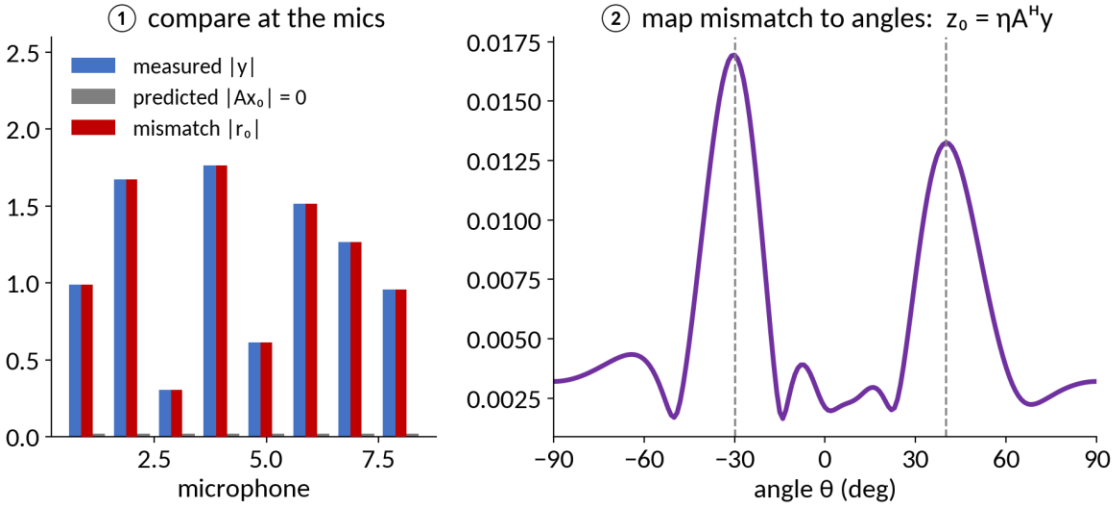
One ISTA iteration, step by step



$$r_0 = A\,x_0 - y$$

Compare with what they did hear: the mismatch $r_0 = Ax_0 - y$ (red).

One ISTA iteration, step by step



$$r_0 = A\,x_0 - y$$

$$z_0 = x_0 - \eta\,A^H(\,A\,x_0 - y\,)$$

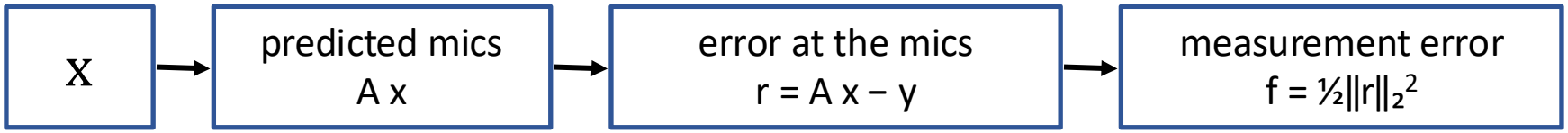
Map the mismatch back to angles with A^H. From $x_0 = 0$ this is exactly the beam scan.

Only now the name: z_0 is a gradient step on $\frac{1}{2}\|Ax - y\|^2$.

Why A^H? Because it is the gradient

$$\nabla_x \tfrac{1}{2}\|Ax - y\|_2^2 = A^H(Ax - y)$$

forward: what the mics would hear



$\nabla_x f = A^H r$ multiply by A^H $\partial f / \partial r = r$

backward: send the mic error back to the angles

$$[A^H r]_k = a(\theta_k)^H r$$

Nudge one source coefficient x_k by δ:
the mics change by $\delta \cdot a(\theta_k)$, so
f changes by $\approx \delta \cdot \mathrm{Re}[\, a(\theta_k)^H r \,]$.

Each component of the gradient says how the measurement error would change if we perturbed one source coefficient x_k.

A^H appears because it is the derivative of the measurement error.

The residual contains the source we missed

Suppose the room has speakers at -30° and $+40^\circ$. Our current estimate x has the -30° speaker right, but misses $+40^\circ$ entirely.

$$y = A\,x + s\,a(40^\circ) + n$$

$$r = A\,x - y \approx -s\,a(40^\circ)$$

The residual is the steering pattern of the missing source, with a minus sign.

Test the residual against every candidate direction

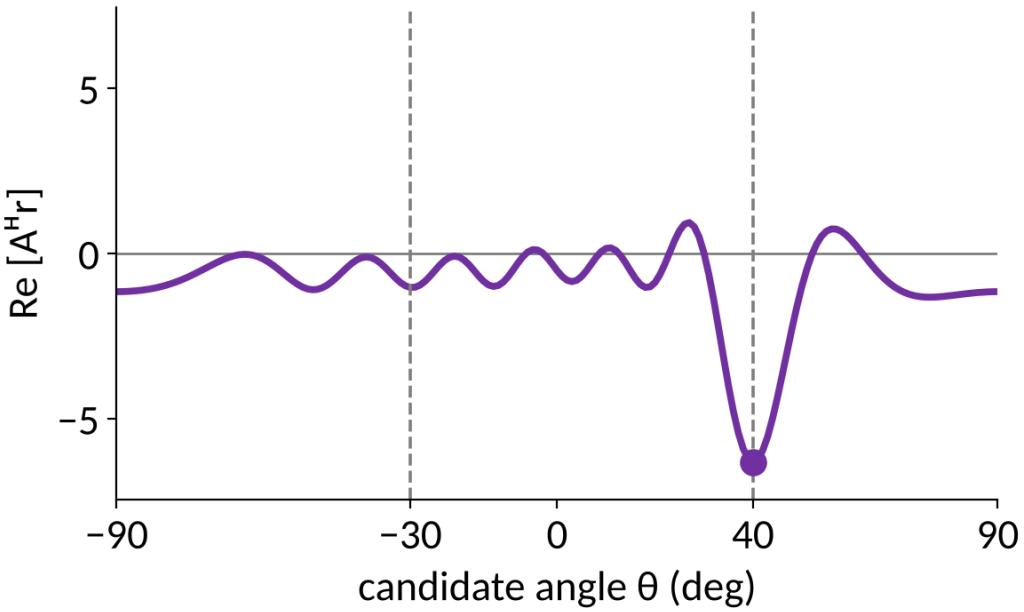
$$A^H r = \begin{bmatrix} | & | & & | \\ a(\theta_1) & a(\theta_2) & \dots & a(\theta_{181}) \\ | & | & & | \end{bmatrix}^H r = \begin{bmatrix} a(\theta_1)^H r \\ a(\theta_2)^H r \\ \vdots \\ a(\theta_{181})^H r \end{bmatrix}$$

one correlation per candidate direction

Test the residual against every candidate direction

$$A^H r = \begin{bmatrix} a(\theta_1)^H r \\ a(\theta_2)^H r \\ \vdots \\ a(\theta_{181})^H r \end{bmatrix}$$

one correlation per candidate direction



At 40° : r looks like a(40°), so a(40°)^{H}r = −6.3 (≈ −8s). Elsewhere the phase patterns do not line up, so the correlation is small.

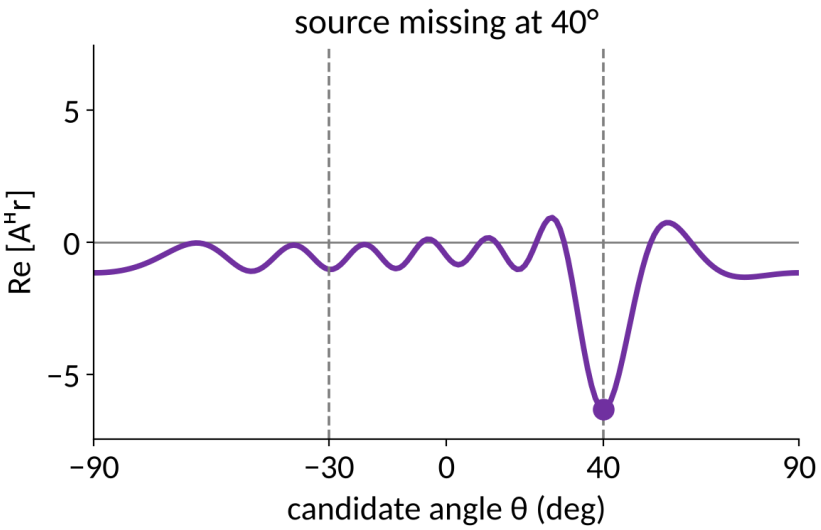
"Which source coefficients are most responsible for the error I currently see at the microphones?"

$A^H r$ scores every candidate direction by how well it explains the error.

The sign tells us which way to correct

$$x_{new} = x - \eta\, A^H r$$

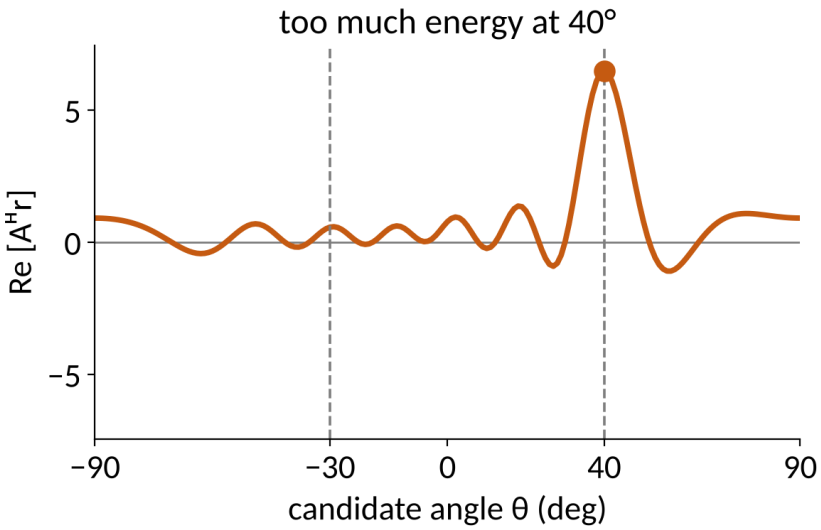
$r \approx -s\, a(40°) \Rightarrow a(40°)^H r < 0$



subtracting a negative raises $x_{40°}$

$x_{40°}$: 0 → +0.014

$r \approx +c\, a(40°) \Rightarrow a(40°)^H r > 0$

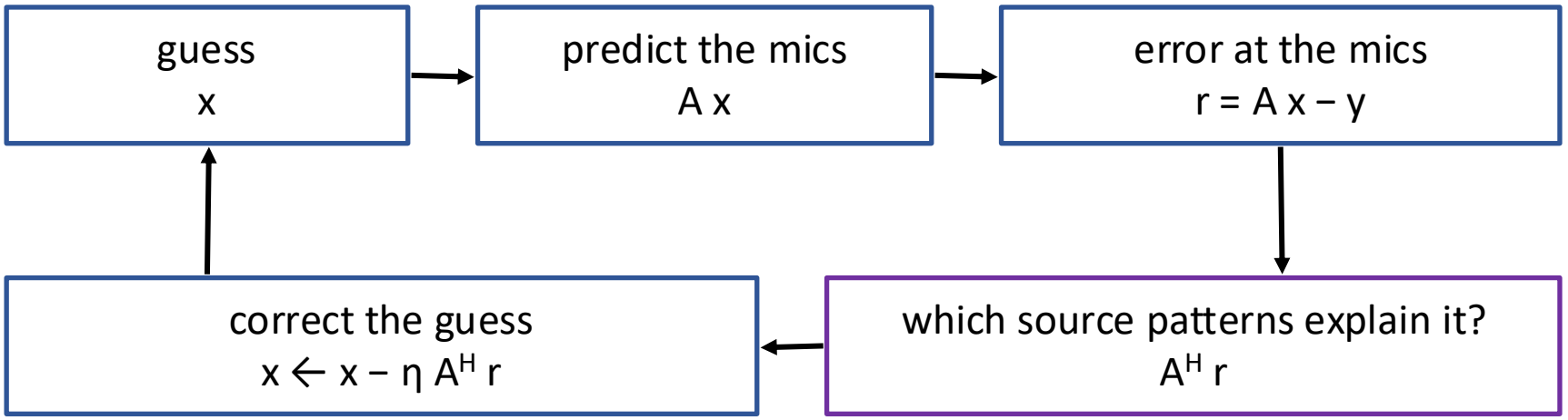


subtracting a positive lowers $x_{40°}$

$x_{40°}$: 1.6 → 1.586

The size of $A^H r$ points at the right direction; its sign says raise or lower.

A physical feedback loop

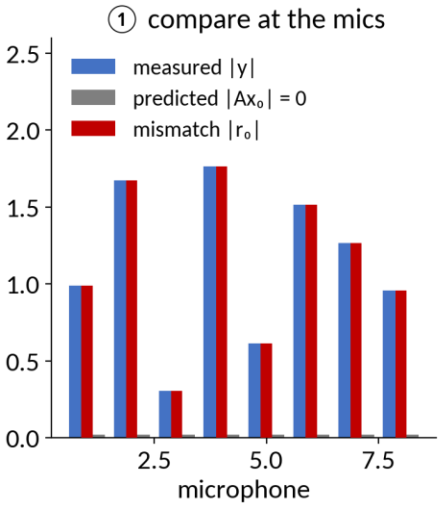


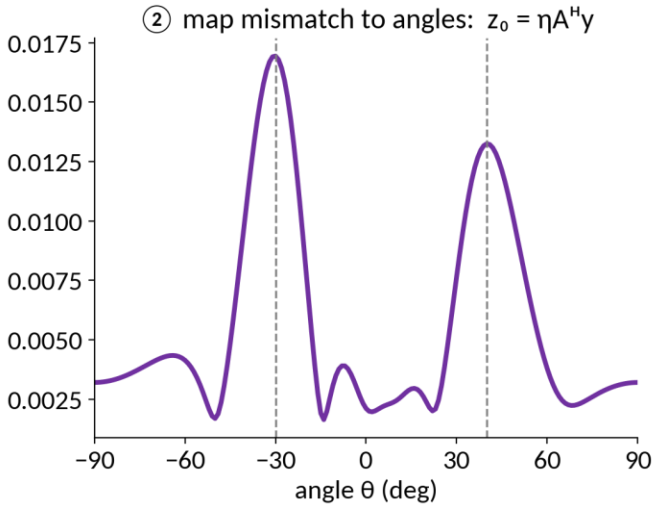
We do not correlate with the residual because it is the desired signal.
We correlate with it because it tells us what our current model still gets wrong.
A^H identifies which source directions could explain that remaining error.

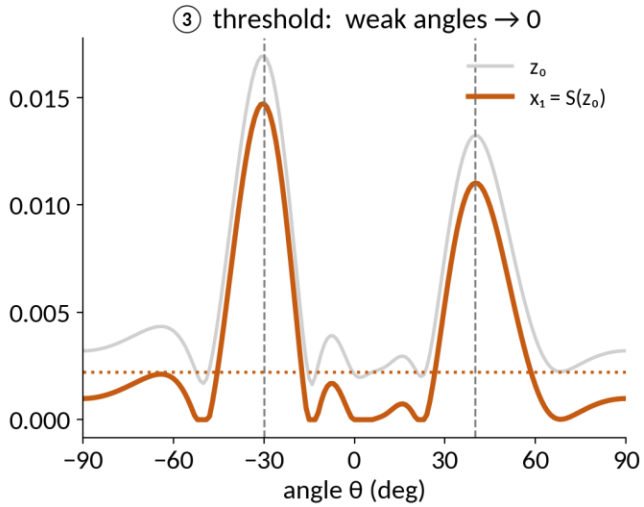
That is exactly why $A^H(Ax - y)$ is the gradient: each component says how the measurement error changes if we perturb one coefficient x_k.

Guess, predict, measure the error, blame the right directions, correct.

One ISTA iteration, step by step







$$x_1 = S_{\eta\lambda}(z_0)$$

Enforce few sources: angles below the threshold are set to 0, the rest shrink.

The threshold enforces the physical sparsity prior.

So $S_{\eta\lambda}$ is **not an arbitrarily chosen activation function**. It is the proximal operator associated with the L_1 regularizer.

$$S_\tau(z) = \begin{cases} z - \tau, & z > \tau \\ 0, & |z| \le \tau \\ z + \tau, & z < -\tau \end{cases} \qquad \tau = \eta\lambda.$$

An equivalent compact form is

$$S_\tau(z) = \operatorname{sign}(z)\max(|z| - \tau, 0).$$

Put the two operations together: ISTA

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta\, A^H(A\, x_k - y)\,]$$

sparsity correction

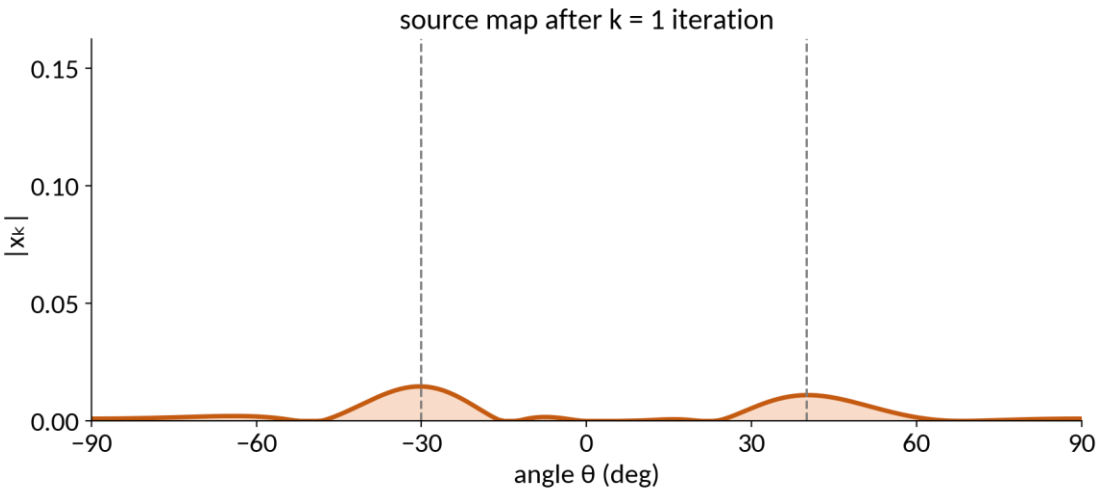
measurement correction

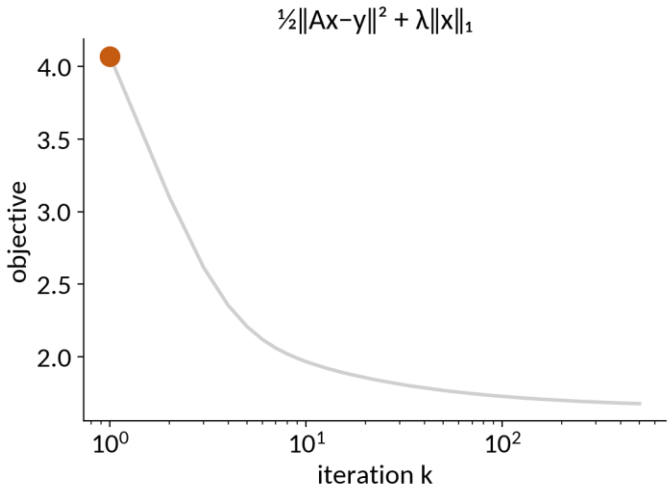
ISTA = Iterative Shrinkage-Thresholding Algorithm

Step size $\eta \le 1/\|A\|_2^2$ guarantees convergence. For our array $\|A\|_2^2 \approx 450$.

ISTA = data consistency + prior enforcement, repeated.

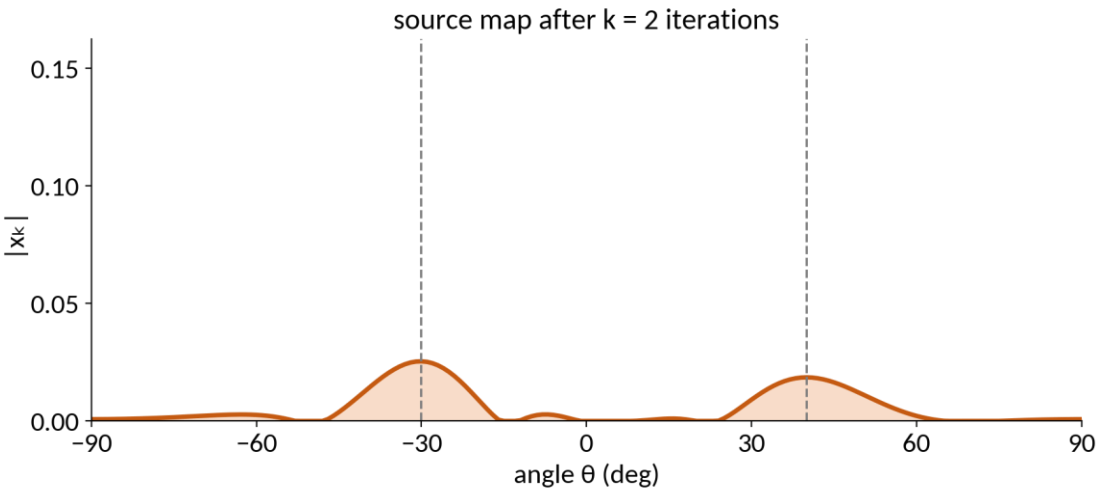
Watch ISTA run: the source map, iteration by iteration

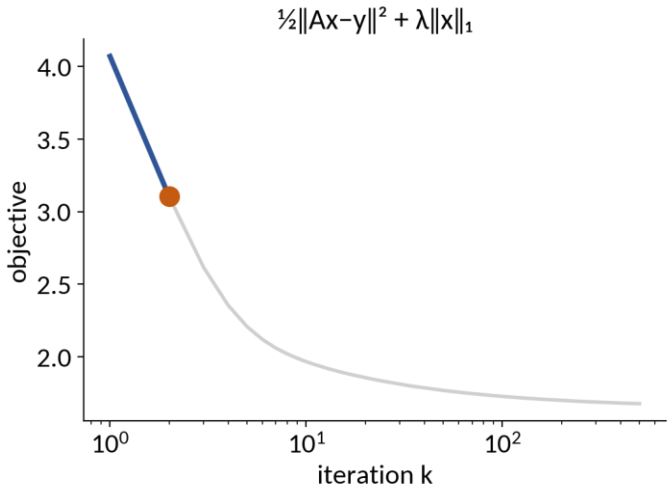




After 1 iteration: a thresholded beam scan.

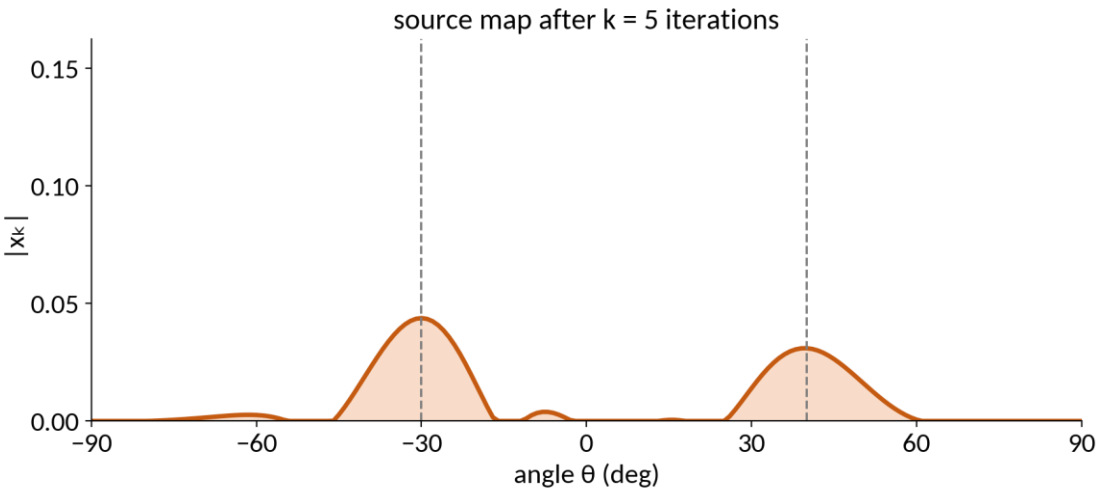
Watch ISTA run: the source map, iteration by iteration

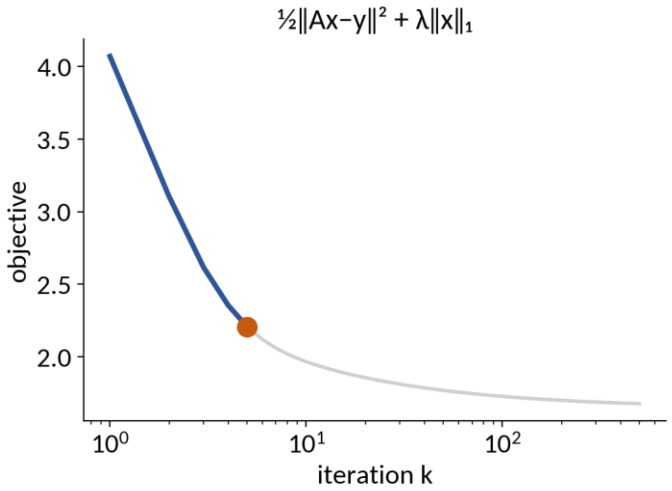




After 2: almost unchanged. Each step is tiny (η = 1/450).

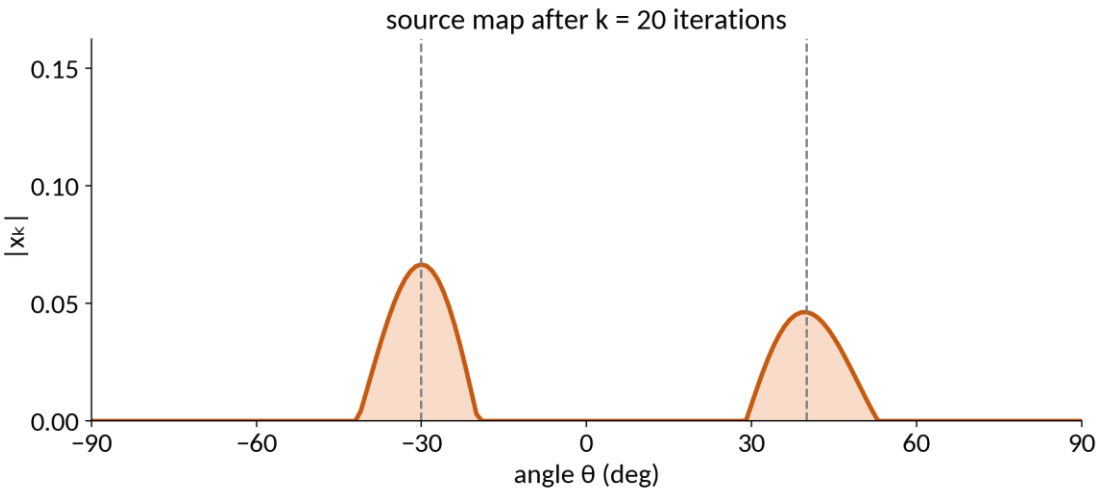
Watch ISTA run: the source map, iteration by iteration

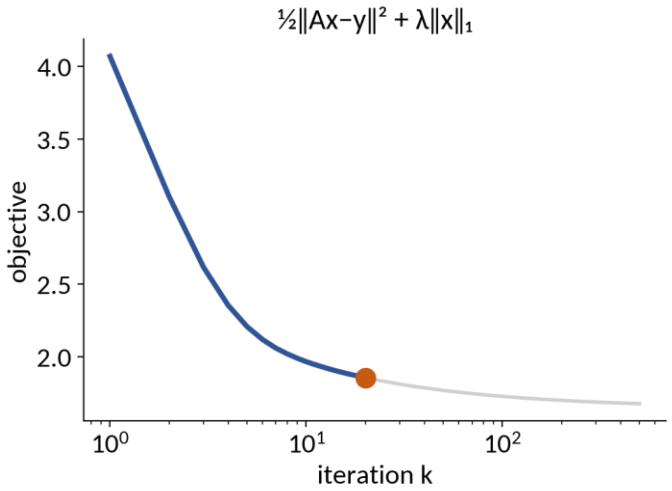




After 5: still broad lobes.

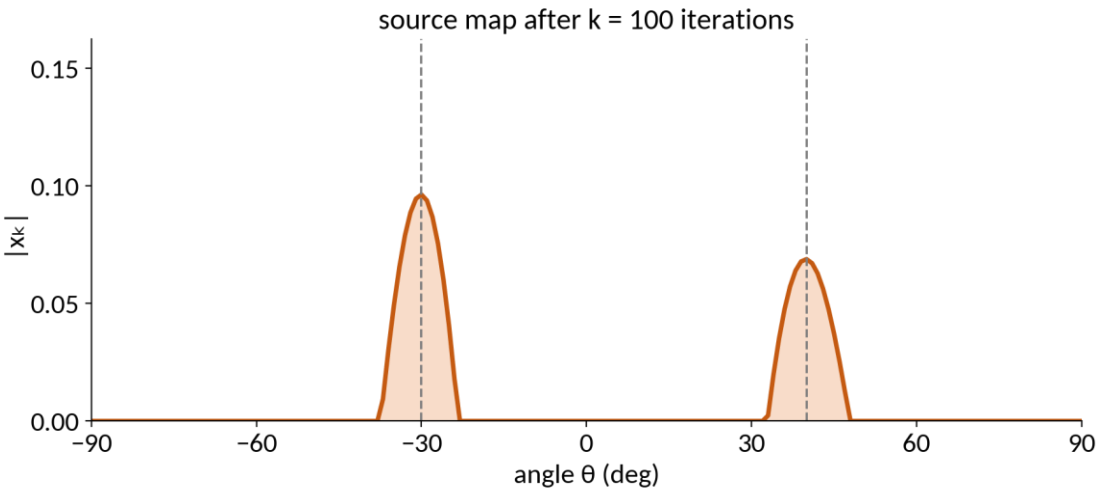
Watch ISTA run: the source map, iteration by iteration

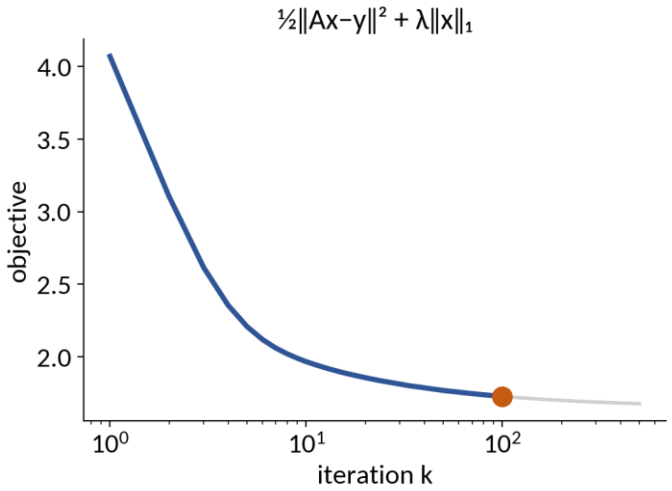




After 20: weak directions start to fade.

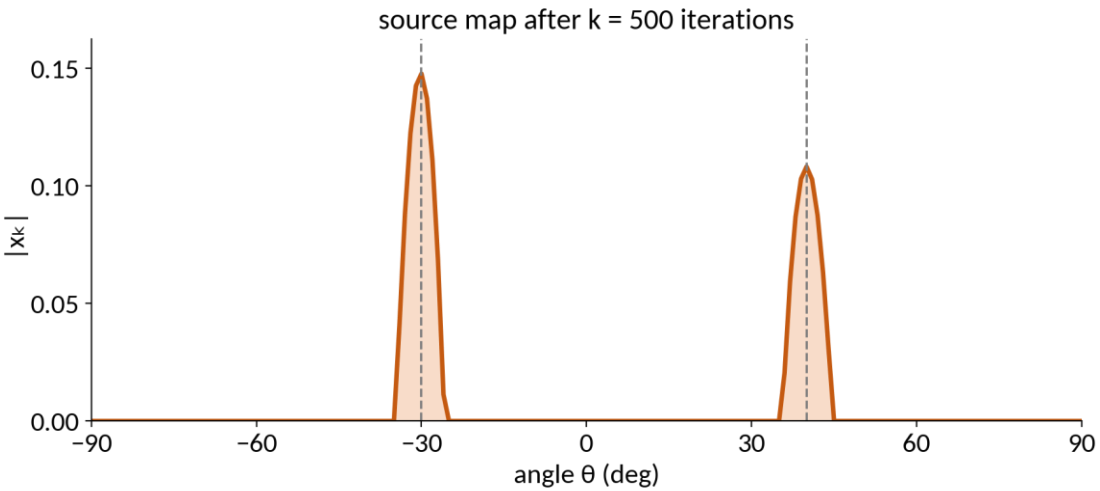
Watch ISTA run: the source map, iteration by iteration

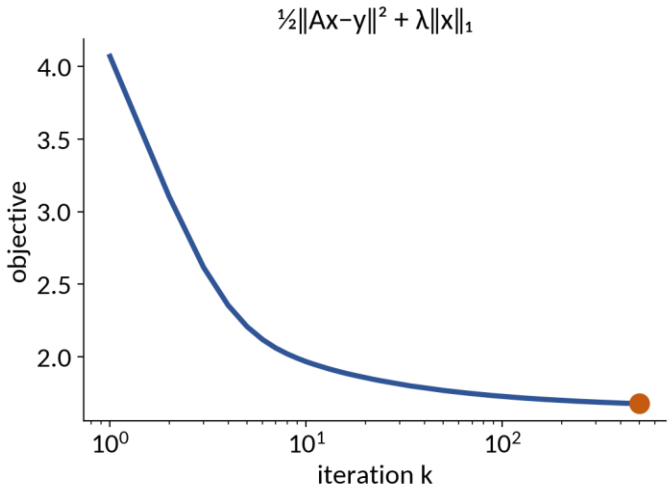




After 100: two narrow peaks emerge.

Watch ISTA run: the source map, iteration by iteration

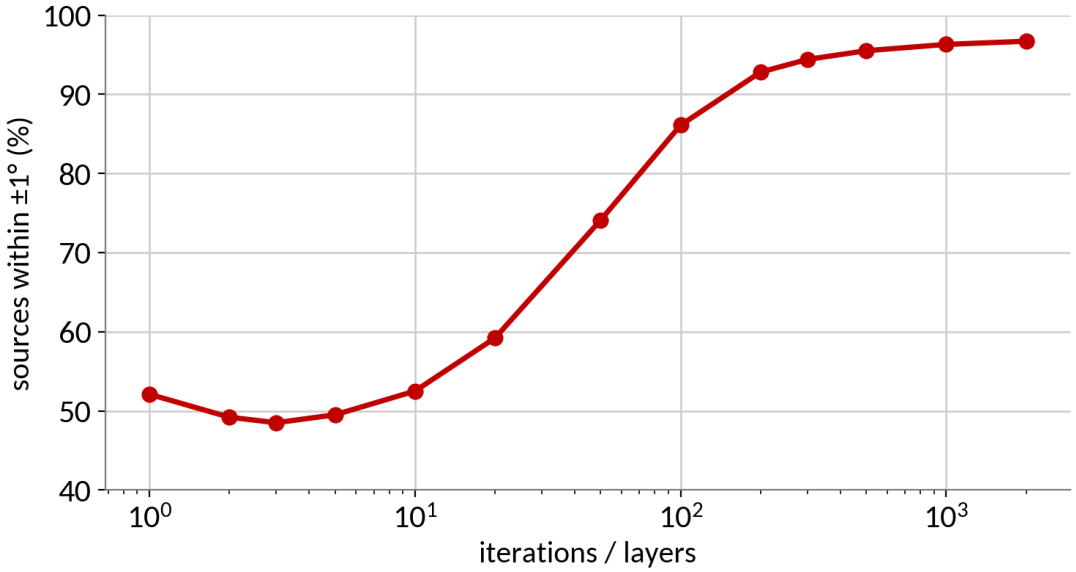




After 500: sharp peaks at the true angles.

If we solve thousands of similar DoA problems, must we always take hundreds of prescribed iterations?

How many iterations does ISTA need?



Iterations	within ±1°
5	49.5%
20	59.2%
100	86.1%
500	95.5%

Small-scale experiment: 4,000 random scenes, 1–2 sources, noise σ = 0.05.

Accuracy = % of true sources whose peak lands within 1°.

Slow convergence and computational cost motivate learning.

Put the two operations together: ISTA

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta\, A^H(A\, x_k - y)\,]$$

sparsity correction

measurement correction

ISTA = Iterative Shrinkage-Thresholding Algorithm

Step size $\eta \le 1/\|A\|_2^2$ guarantees convergence. For our array $\|A\|_2^2 \approx 450$.

ISTA = data consistency + prior enforcement, repeated.

How does L1 regularization become an activation function?

SPARSE INVERSE PROBLEM

$$\min_x \underbrace{\tfrac{1}{2}\|Ax - y\|_2^2}_{\text{data fidelity}} + \lambda \underbrace{\|x\|_1}_{\text{sparsity}}$$

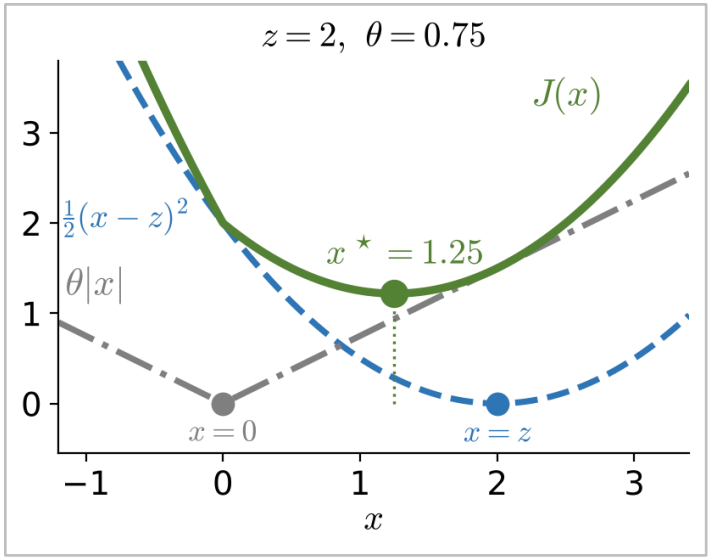
ISTA handles the two terms separately:

① gradient $z_k = x_k - \eta A^T(Ax_k - y)$

② proximal $x_{k+1} = \arg\min_x [\, \tfrac{1}{2}\|x - z_k\|^2 + \eta\lambda\|x\|_1 \,]$

SEPARABLE: SOLVE ONE COEFFICIENT

$$\min_x J(x) = \tfrac{1}{2}(x - z)^2 + \theta|x|, \quad \theta = \eta\lambda$$



quadratic → z, penalty → 0, the sum balances them

Case 1: x > 0

$J'(x) = x - z + \theta = 0$

$\Rightarrow\ x = z - \theta$

valid when $z > \theta$

Case 2: x < 0

$J'(x) = x - z - \theta = 0$

$\Rightarrow\ x = z + \theta$

valid when $z < -\theta$

Case 3: x = 0

$0 \in -z + \theta[-1, 1]$

$\Rightarrow\ |z| \le \theta$

subgradient of $|x|$ at 0 is $[-1, 1]$

$$S_\theta(z) = \begin{cases} z - \theta & z > \theta \\ 0 & |z| \le \theta \\ z + \theta & z < -\theta \end{cases}$$

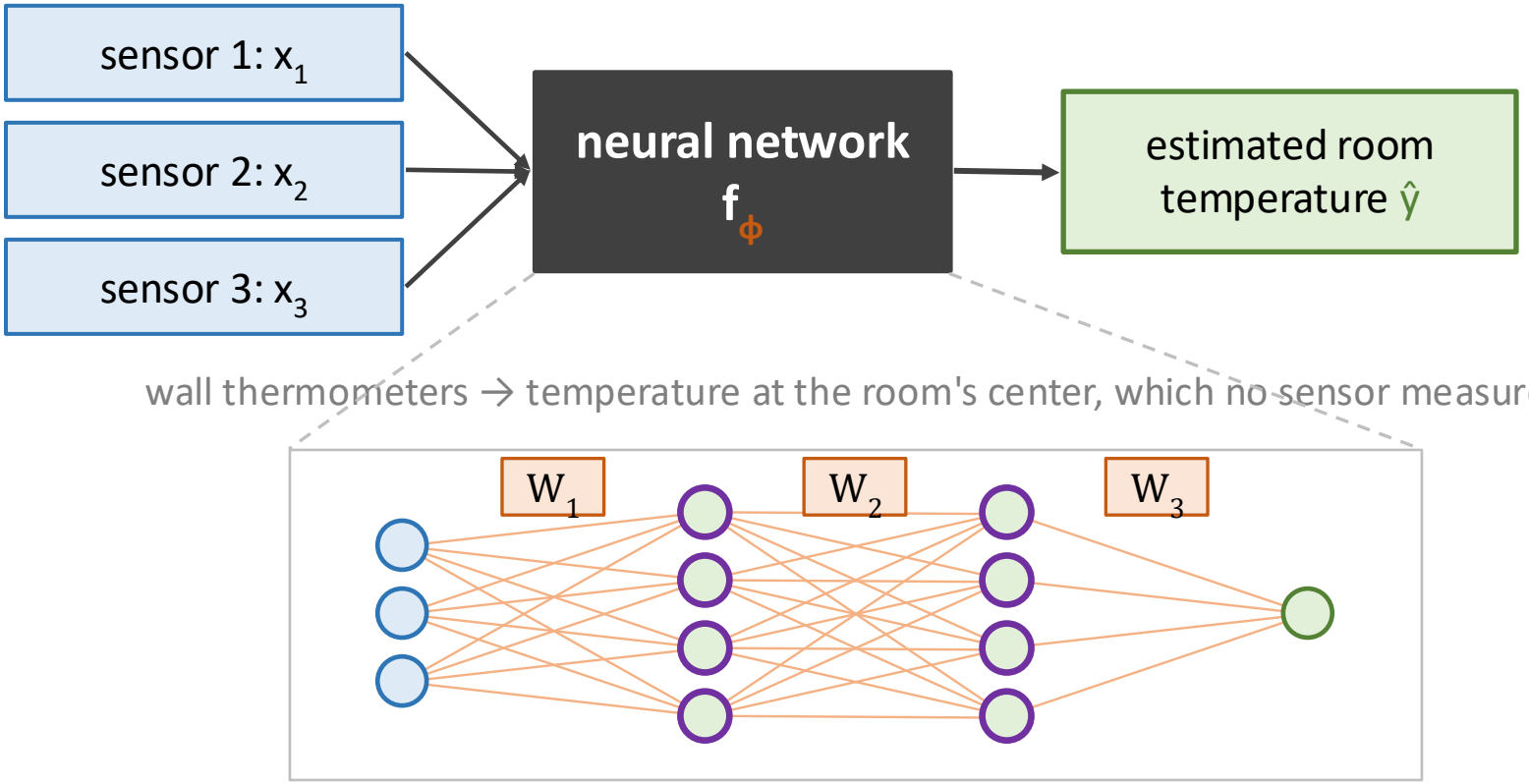
Soft-thresholding is derived from the L1 penalty, not chosen.

Neural Network Basics

A neural network is a learnable mathematical function

$$\hat{y} = f_{\varphi}(x)$$

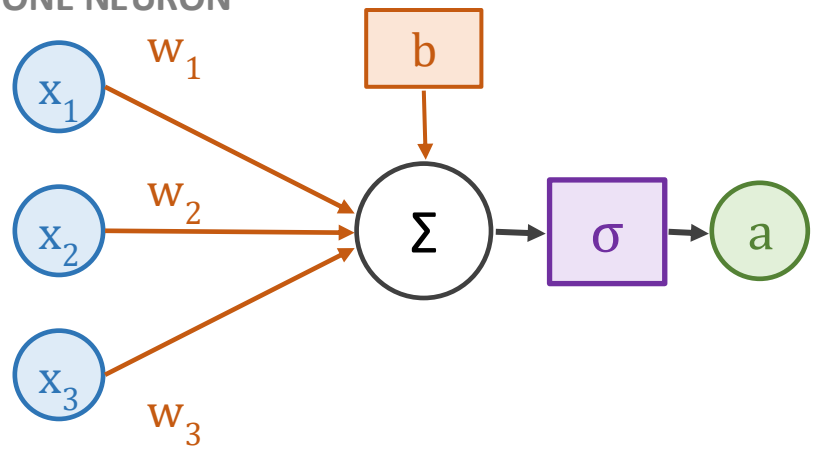
x: input data ŷ: predicted output ϕ: trainable parameters



Not a black box: a composition of math operations with trainable parameters ϕ.

Inside the network: weighted transformations

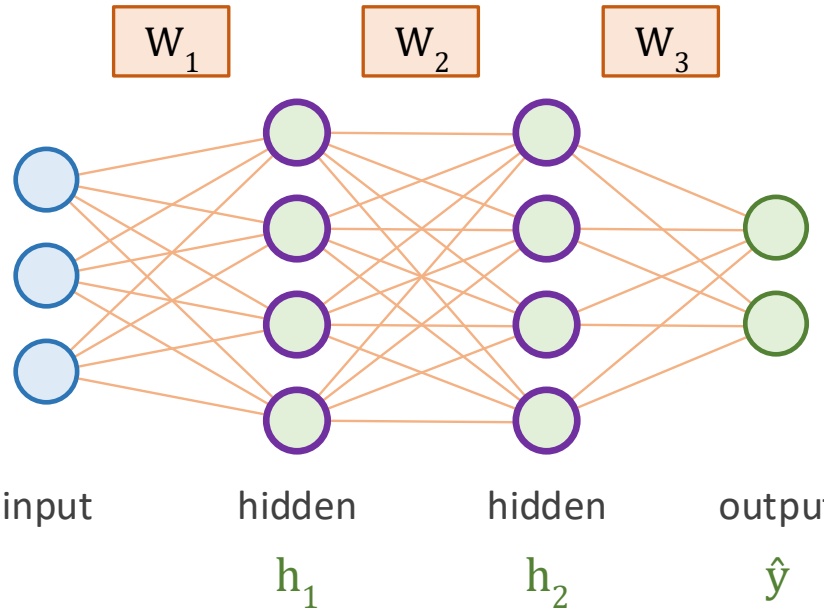
ONE NEURON



$$z = \Sigma_i \, w_i x_i + b$$

$$a = \sigma(z)$$

A WHOLE NETWORK



$$h_{k+1} = \sigma(W_k \, h_k + b_k)$$

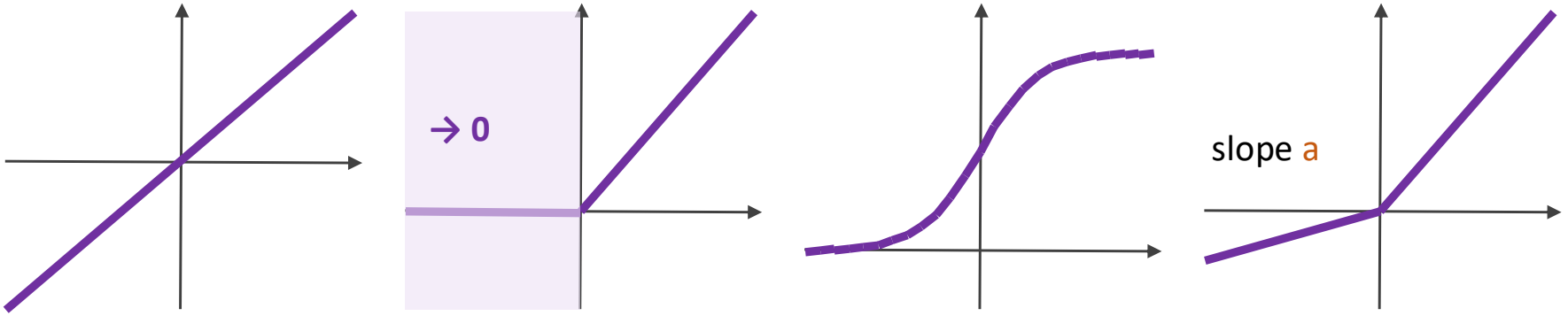
W_k, b_k: trainable parameters, fixed after training. h_k: intermediate values, recomputed for every input.

Each layer: a trainable matrix, a bias, then a nonlinearity.

Nonlinearity makes neural networks powerful

$$W_2(W_1 x) = (W_2 W_1)x$$

without σ, two layers collapse into one linear map



linear: z | $\mathrm{ReLU}(z) = \max(0, z)$ | sigmoid: $1 / (1 + e^{-z})$ | PReLU: z, or az ($z < 0$)

ReLU zeroes negative inputs and keeps positive ones. tanh is like sigmoid with range (−1, 1).

In PReLU the slope a is a trainable parameter, learned by gradient descent like any weight.

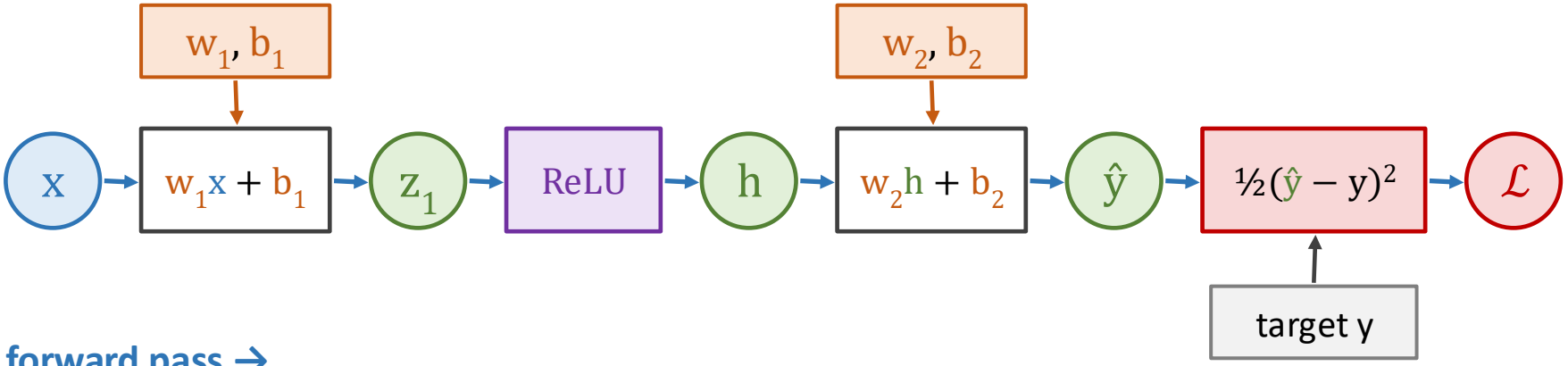
Activations add nonlinearity; some, like PReLU, have trainable parameters.

How does the network produce a prediction?

$$h = \mathrm{ReLU}(w_1 x + b_1)$$

$$\hat{y} = w_2 h + b_2$$

$$\mathcal{L} = \tfrac{1}{2}(\hat{y} - y)^2$$



forward pass →

example: $x = 2,\ y = 3,\ w_1 = w_2 = 1,\ b_1 = b_2 = 0$

x	z_1	h	$\hat{y}$	$\mathcal{L}$
2	2	2	2	0.5

The forward pass computes a prediction; the loss measures its error.

Learning means changing parameters to reduce loss

$$\hat{y} = wx + b$$

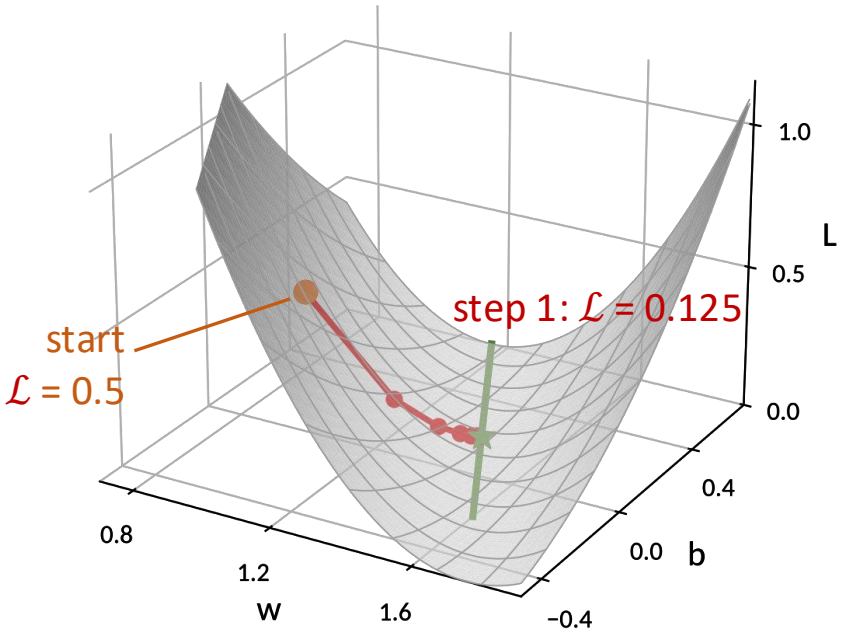
$$\mathcal{L} = \tfrac{1}{2}(wx + b - y)^2$$

$$\partial\mathcal{L}/\partial w = (wx + b - y)\, x$$

$$\partial\mathcal{L}/\partial b = wx + b - y$$

$$w_{new} = w - \alpha\, \partial\mathcal{L}/\partial w$$

α: learning rate (same rule for b)



$x = 2, y = 3$
$w = 1, b = 0$
$\hat{y} = 2$

→

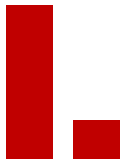
$\partial\mathcal{L}/\partial w = -2$
$\partial\mathcal{L}/\partial b = -1$

→

$\alpha = 0.1$
$w \to 1.2$
$b \to 0.1$

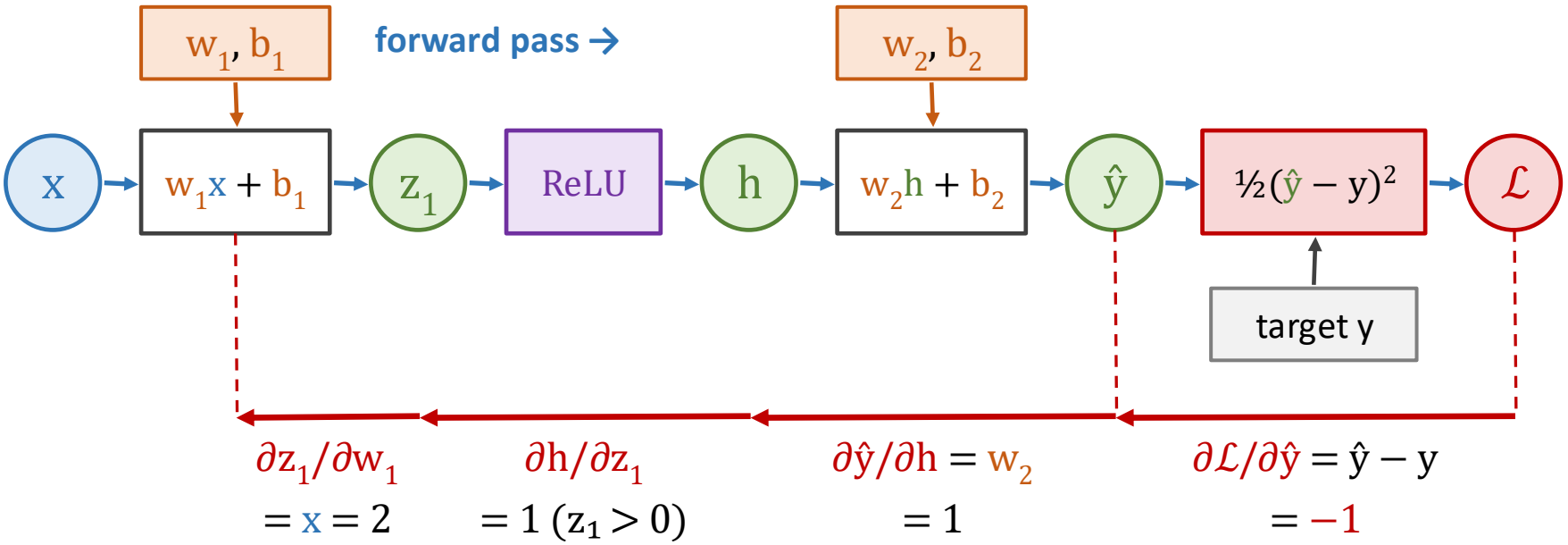
→

$\hat{y} = 2.5$
$\mathcal{L} = 0.125$



Differentiation tells us how changing each parameter changes the loss.

How does the error reach earlier layers?



$$\partial \mathcal{L}/\partial w_1 = \partial \mathcal{L}/\partial \hat{y} \cdot \partial \hat{y}/\partial h \cdot \partial h/\partial z_1 \cdot \partial z_1/\partial w_1$$
$$= (-1)(1)(1)(2) = -2$$

$$\partial \mathcal{L}/\partial w_2 = \partial \mathcal{L}/\partial \hat{y} \cdot h = (-1)(2) = -2 \qquad \partial \mathcal{L}/\partial b_1 = \partial \mathcal{L}/\partial b_2 = -1$$

Backpropagation computes every parameter's gradient in one backward sweep.

Computing gradients and updating weights are different jobs

	Backpropagation	Gradient descent
job	computes derivatives	updates parameters
uses	the chain rule	the computed gradient
produces	$\nabla_\varphi \mathcal{L}$	$\varphi \leftarrow \varphi - \alpha \nabla_\varphi \mathcal{L}$
changes the weights?	no	yes

backpropagation gives

$$\nabla_\varphi \mathcal{L} \text{ for } (w_1, b_1, w_2, b_2):$$
$$= (-2, -1, -2, -1)$$

Nothing has changed yet: the weights are still 1, 0, 1, 0.

gradient descent applies it ($\alpha = 0.1$)

w_1: $1 \rightarrow 1.2$ $\quad$ b_1: $0 \rightarrow 0.1$

w_2: $1 \rightarrow 1.2$ $\quad$ b_2: $0 \rightarrow 0.1$

new $\hat{y} = 3.1$, $\quad \mathcal{L}$: $0.5 \rightarrow 0.005$

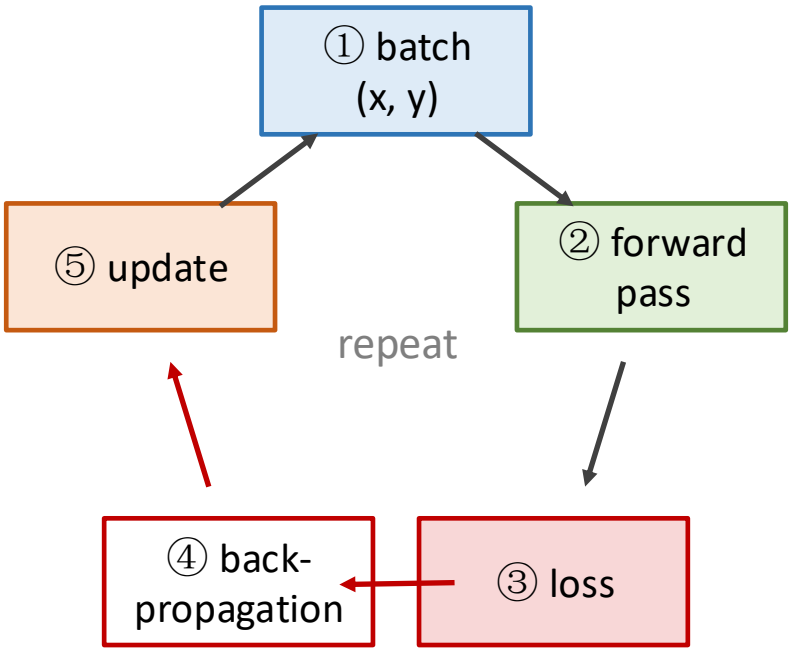
Backprop finds how $\mathcal{L}$ depends on each parameter; the optimizer changes them.

What happens during one training step?

```
for x, y in dataloader:
    optimizer.zero_grad()

    prediction = model(x)
    loss = criterion(prediction, y)

    loss.backward()
    optimizer.step()
```



⓪ zero_grad clears last step's gradients (PyTorch adds them up by default)

batch	training step	epoch	inference
a group of training examples	one parameter update	one full pass over the training set	forward pass only, parameters fixed

Training repeats forward, loss, backward, update, over many batches.

From ISTA to LISTA

Turn the solver into an architecture, then learn it

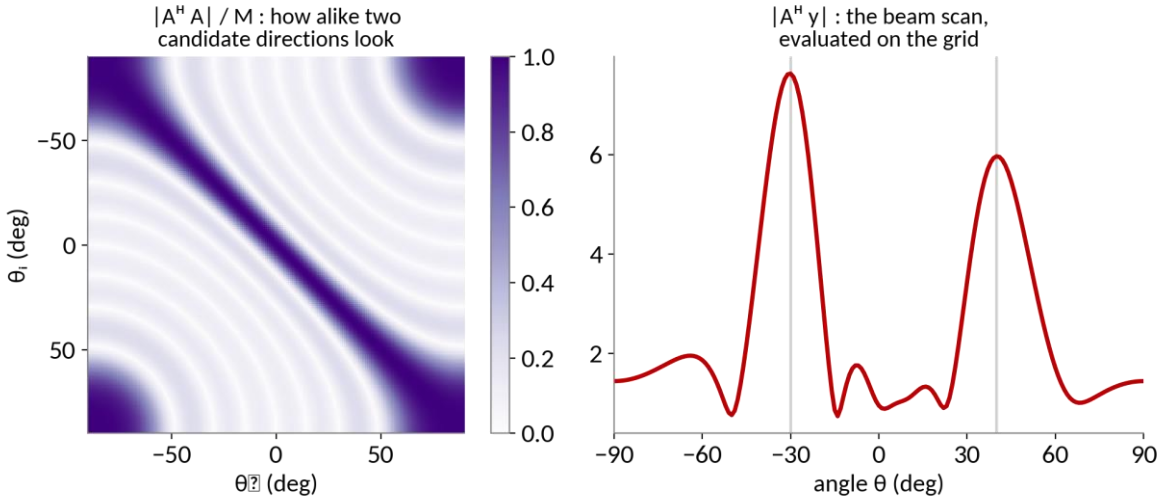
Rewrite ISTA so its structure is visible

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta\, A^H(A\, x_k - y)\,]$$

$$x_{k+1} = S_{\eta\lambda}[\, (I - \eta A^H A)\, x_k + \eta A^H\, y\,]$$

$W_s = I - \eta A^H A \qquad W_e = \eta A^H \qquad \theta = \eta\lambda$

$$x_{k+1} = S_{\theta}(\, W_s\, x_k + W_e\, y\,)$$



Why learn?

Parameter	In ISTA	In LISTA	Potential benefit
W_e	Fixed at ηA^H	Trainable matrix	Learn a better transformation from measurements to candidate signal coefficients
W_s	Fixed at $I - \eta A^H A$	Trainable matrix	Learn how to correct or suppress interference from previous estimates
θ	Fixed at $\eta\lambda$	Trainable threshold	Learn how aggressively to suppress small coefficients
Number of layers K	Iterations until a stopping condition	Fixed unrolled depth	Predictable inference cost with potentially far fewer iterations

The learning question

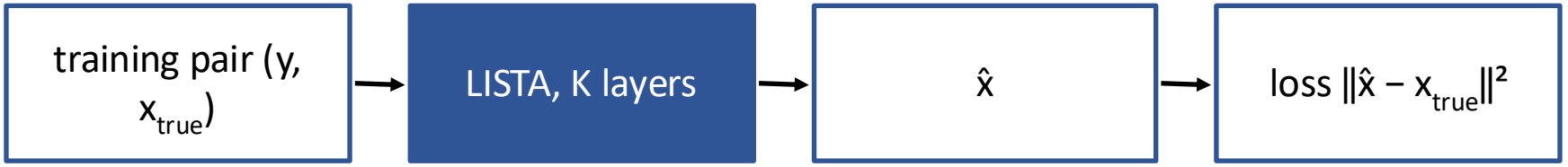
	ISTA (prescribed)	LISTA (learned)
evidence map	$W_e = \eta A^H$	$W_{e,k}$ per layer
state map	$W_s = I - \eta A^H A$	$W_{s,k}$ per layer
threshold	$\theta = \eta\lambda$	θ_k per layer
depth	hundreds of iterations	K = 5 layers

$$\text{LISTA:}\quad x_{k+1} = S_{\theta_\{k\}}\left(W_{s,k}\, x_k + W_{e,k}\, y \right)$$

Can training examples teach parameters that reach an accurate DoA map in far fewer layers?

LISTA learns the update rule but keeps the template of sparse reconstruction.

What is trained, and what is reconstructed?



backpropagation updates $W_{e,k}$, $W_{s,k}$, ϑ_k across many examples

x: reconstructed

- the source map for one snapshot
- recomputed for every new y
- changes layer by layer

W, θ: trained

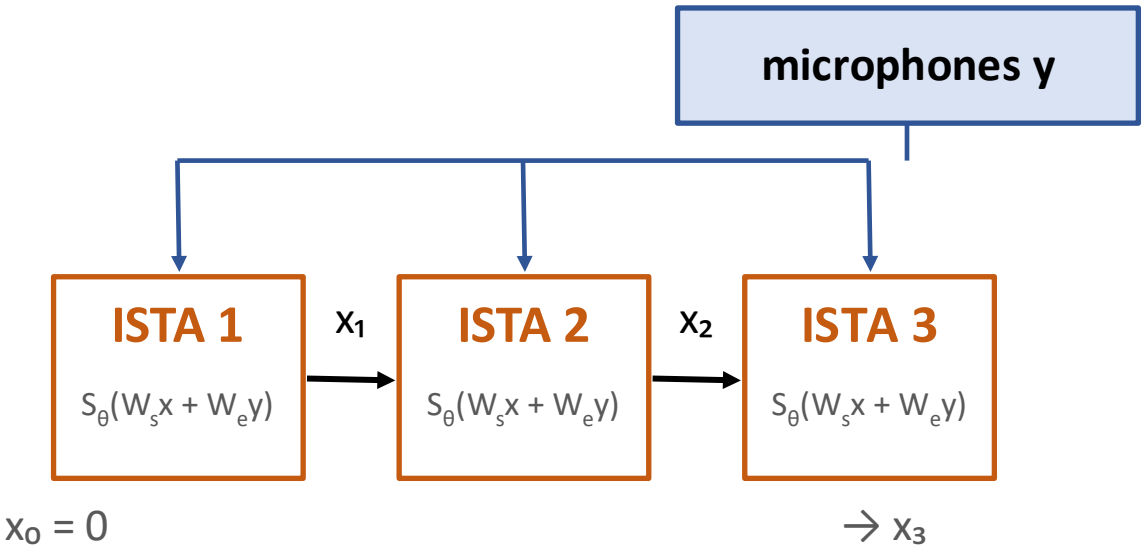
- parameters of the solver
- learned once, across the dataset
- frozen at test time

x is inferred per measurement; W and θ are learned across the dataset.

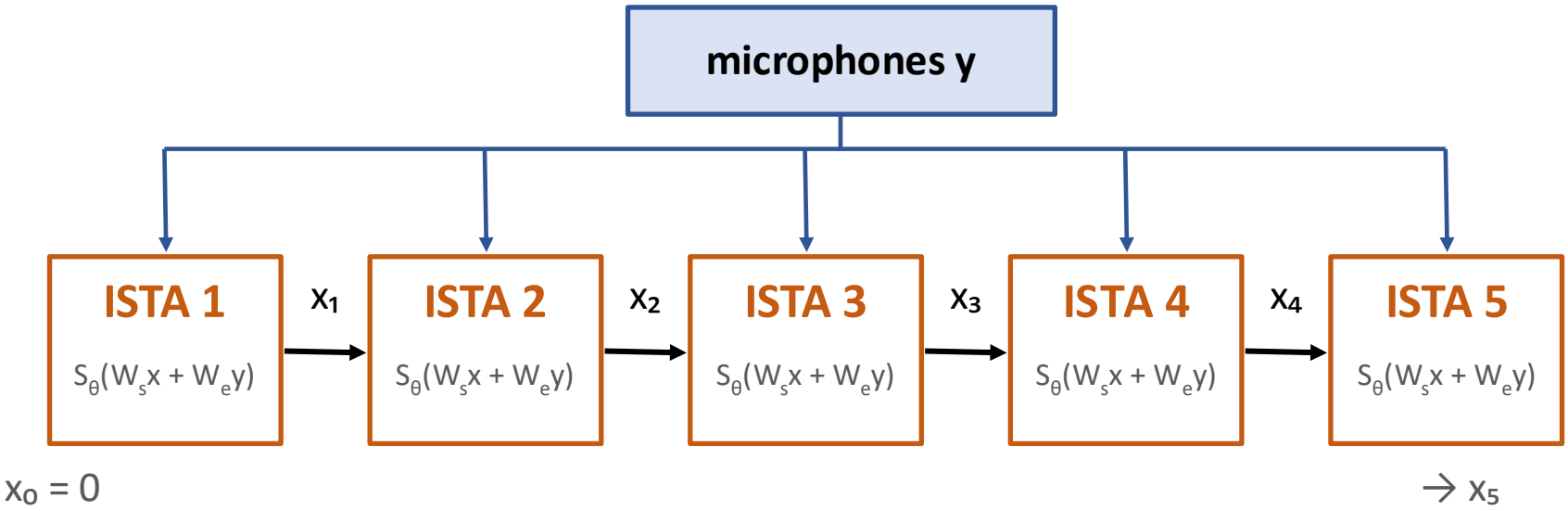
Unroll a fixed number of iterations



Unroll a fixed number of iterations



Unroll a fixed number of iterations

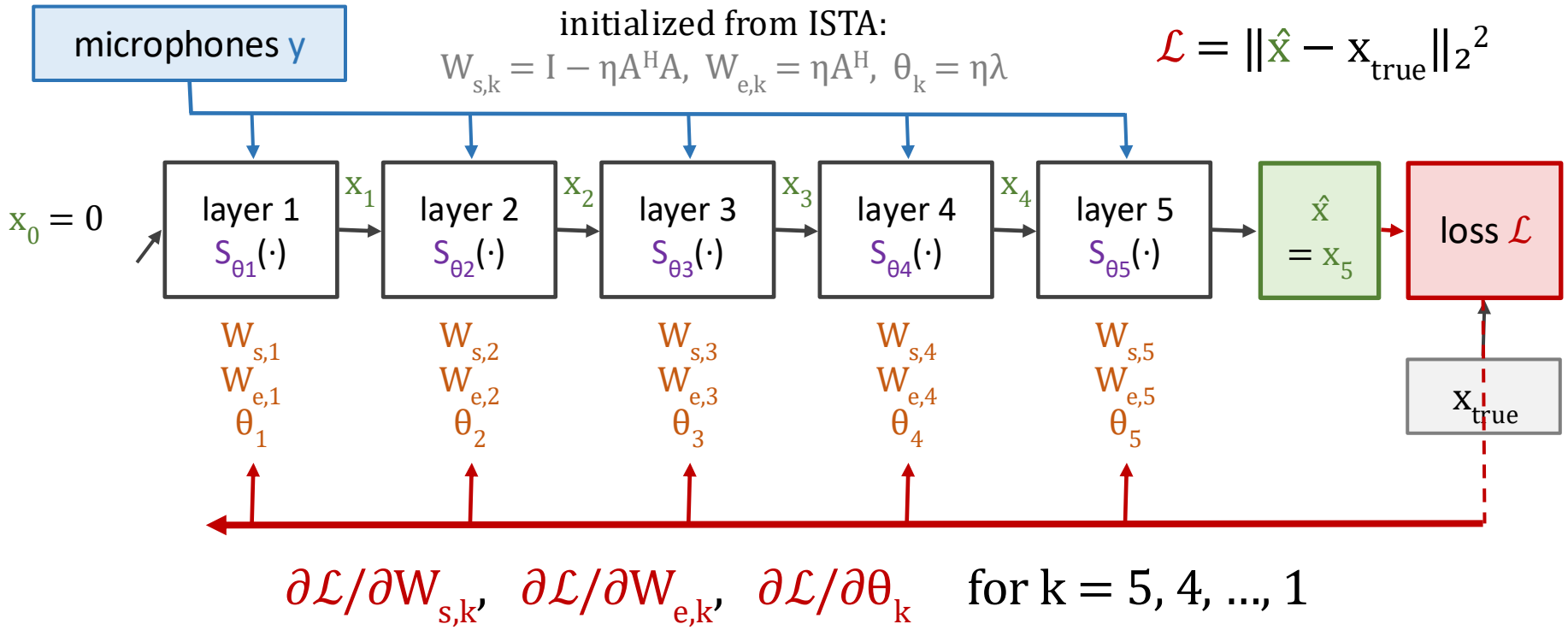


Nothing neural has happened yet. We have only drawn K iterations as a computation graph.

Every block uses the same W_s, W_e, θ, fixed by A, η and λ.

Unrolling turns an iterative algorithm into a finite-depth architecture.

How does LISTA learn its weights?



$\partial\mathcal{L}/\partial W_{s,k}$, $\partial\mathcal{L}/\partial W_{e,k}$, $\partial\mathcal{L}/\partial\theta_k$ for $k = 5, 4, \ldots, 1$

Training: $W_{s,k}$, $W_{e,k}$, θ_k change after every batch, by an optimizer step

Inference: W, θ frozen; $x_1 \ldots x_5$ recomputed for each new y

No extra network makes the matrices: they are trainable tensors, like MLP weights.

Two gradients, two different jobs

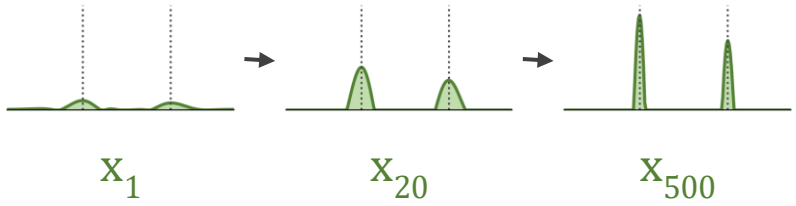
Inside ISTA: gradient w.r.t. x

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta \nabla_x f(x_k)\,]$$

$$f(x) = \tfrac{1}{2}\|Ax - y\|_2^2$$

What changes: the source estimate x

Purpose: solve the inverse problem for one snapshot y



one y, many steps; nothing is learned

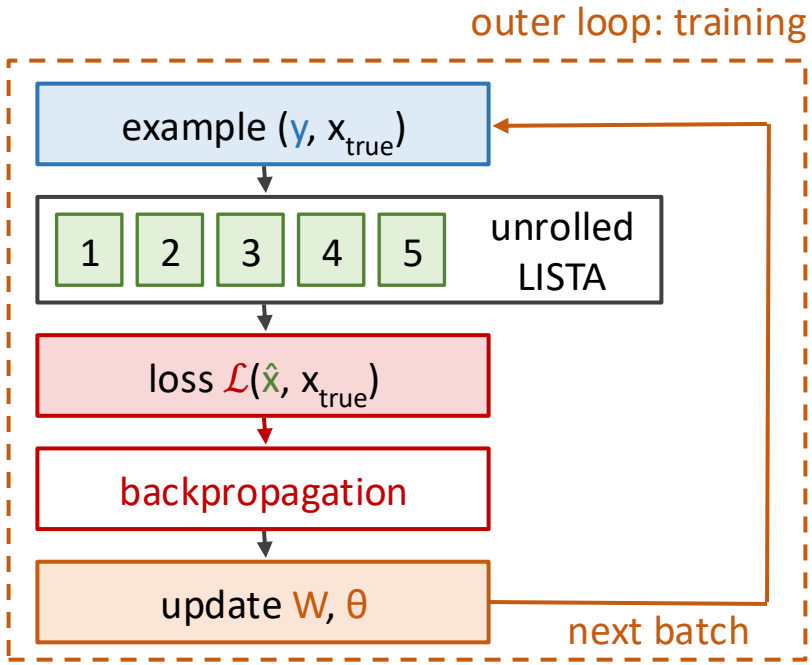
η = ISTA step size (fixed)

Training LISTA: gradient w.r.t. W, θ

$$W \leftarrow W - \alpha \nabla_W \mathcal{L}$$

What changes: $W_{s,k}$, $W_{e,k}$, θ_k

Purpose: improve the solver across a dataset

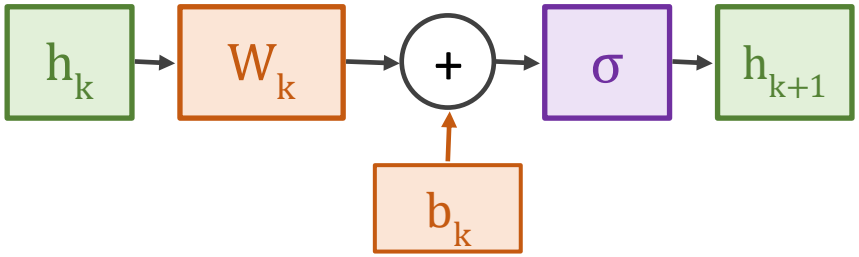


ISTA optimizes a source estimate; training LISTA optimizes the procedure.

A neural network does not have to look like an MLP

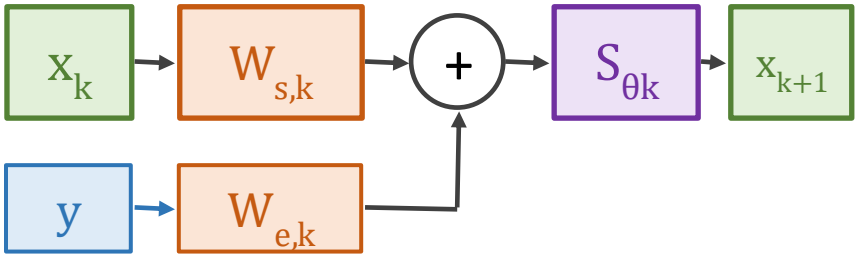
Conventional MLP layer

$$h_{k+1} = \sigma(W_k h_k + b_k)$$



LISTA layer

$$x_{k+1} = S_{\theta k}(W_{s,k} x_k + W_{e,k} y)$$



In LISTA the roles swap: y is the measurement (input), x the estimate.

Shared
trainable weight matrices
a nonlinear activation
trained by backpropagation

LISTA-specific
x_k: iterative source estimates
measurements y enter every layer
structure from an optimization algorithm

A neural network: any parameterized computation whose parameters are learned.

Back to our microphone array: can we directly train an MLP?

8 microphones · 181 candidate directions · 1–2 speakers



$y \in \mathbb{C}^8$: one snapshot

$[\ \mathrm{Re}\ y\ ;\ \mathrm{Im}\ y\ ] \in \mathbb{R}^{16}$

↓

dense W_1, then ReLU

↓

dense W_2, then ReLU

↓

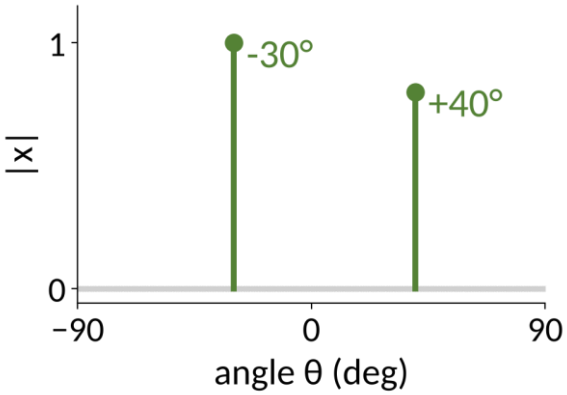
dense W_3, then ReLU

↓

dense W_4: 2 → 181 outputs

↓

$$\hat{x} = \mathrm{MLP}(y)$$

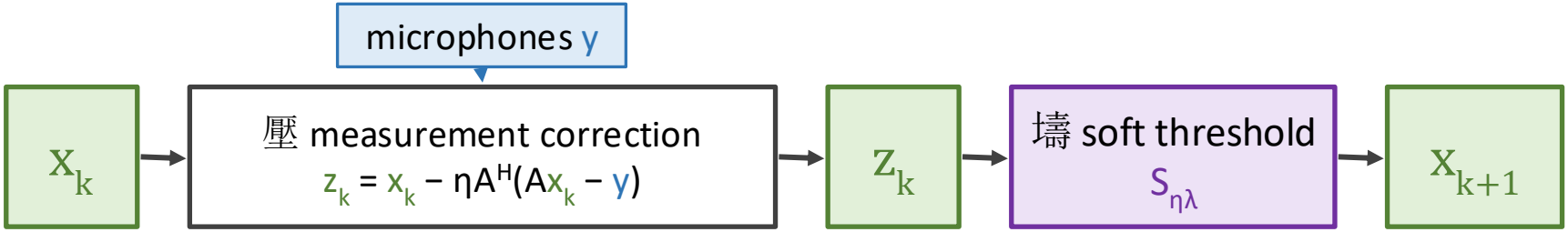


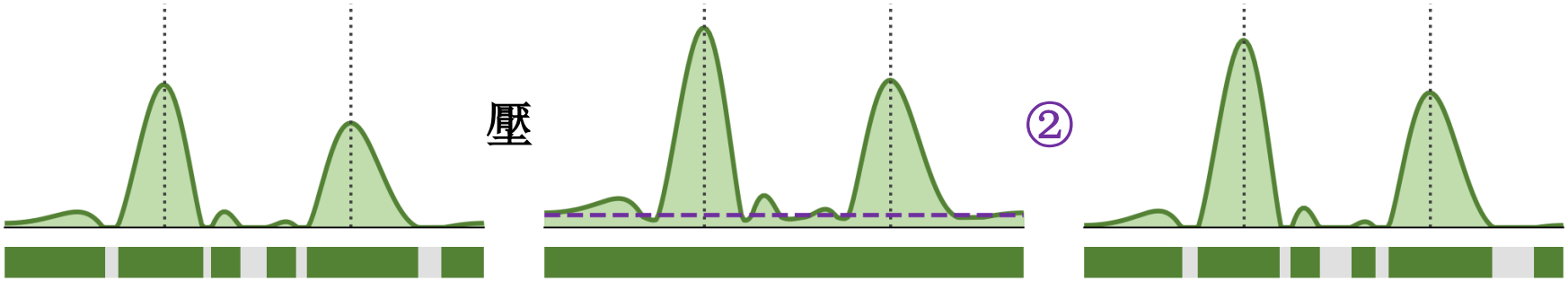
desired $x \in \mathbb{C}^{181}$: two peaks

? If an MLP can learn the mapping y → x, why do we need LISTA?

We already have an algorithm that knows how to solve this

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta\, A^H(A\, x_k - y)\,]$$





x_2: 150 of 181 angles active
mismatch $\|Ax - y\|/\|y\| = 59\%$

z_2: all 181 active
mismatch 40% (better fit)

x_3: 138 active
mismatch 48% (weak angles cut)

Real ISTA step on our two-speaker snapshot. Dotted: true speakers. Bar under each map: green = angle kept. Purple dashed: threshold $\eta\lambda$.

ISTA already has a meaningful structure. Can that structure learn?

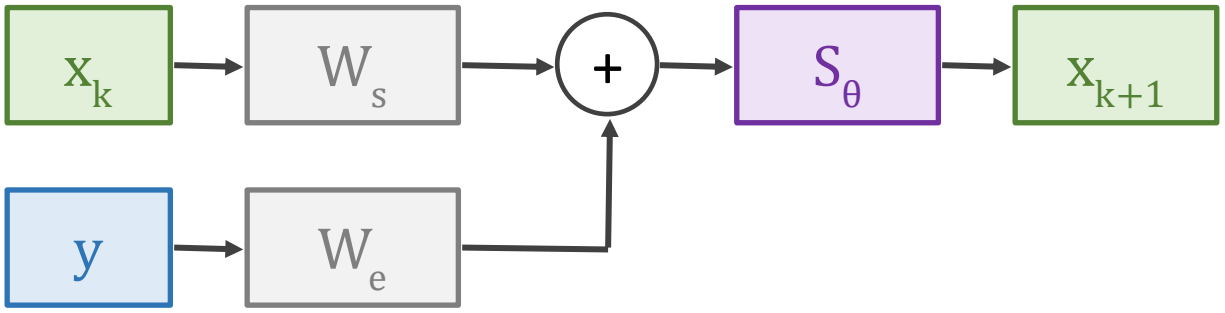
An ISTA iteration already looks like a neural-network layer

$$x_{k+1} = S_{\eta\lambda}[\, x_k - \eta A^H(Ax_k - y)\,]$$
$$= S_{\eta\lambda}[\, (I - \eta A^H A)x_k + \eta A^H y\,]$$

$$W_s = I - \eta A^H A \qquad W_e = \eta A^H \qquad \theta = \eta\lambda$$

gray = fixed: computed from A, η, λ, not learned

$$x_{k+1} = S_\theta(\, W_s\, x_k + W_e\, y\,)$$



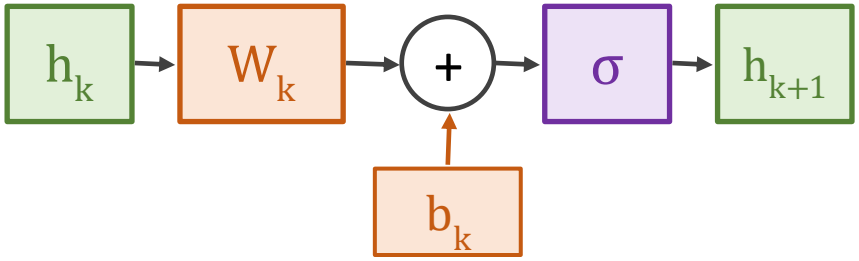
$W_s x_k$: current estimate
$W_e y$: measurements
+: combine them
S_θ: nonlinearity

The anatomy of a layer, but every number is derived, not learned.

MLP layer vs. LISTA layer: more similar than they appear

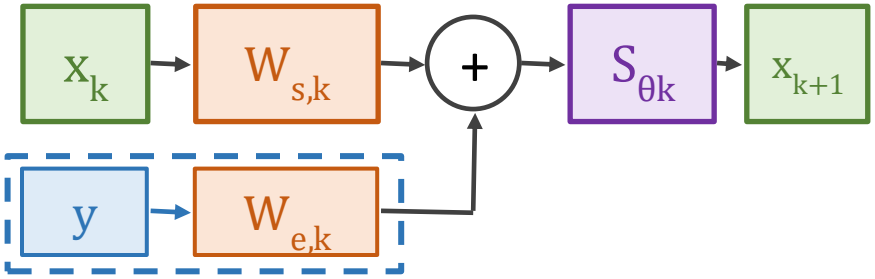
MLP layer

$$h_{k+1} = \sigma(W_k h_k + b_k)$$



LISTA layer

$$x_{k+1} = S_{\theta k}(W_{s,k} x_k + W_{e,k} y)$$



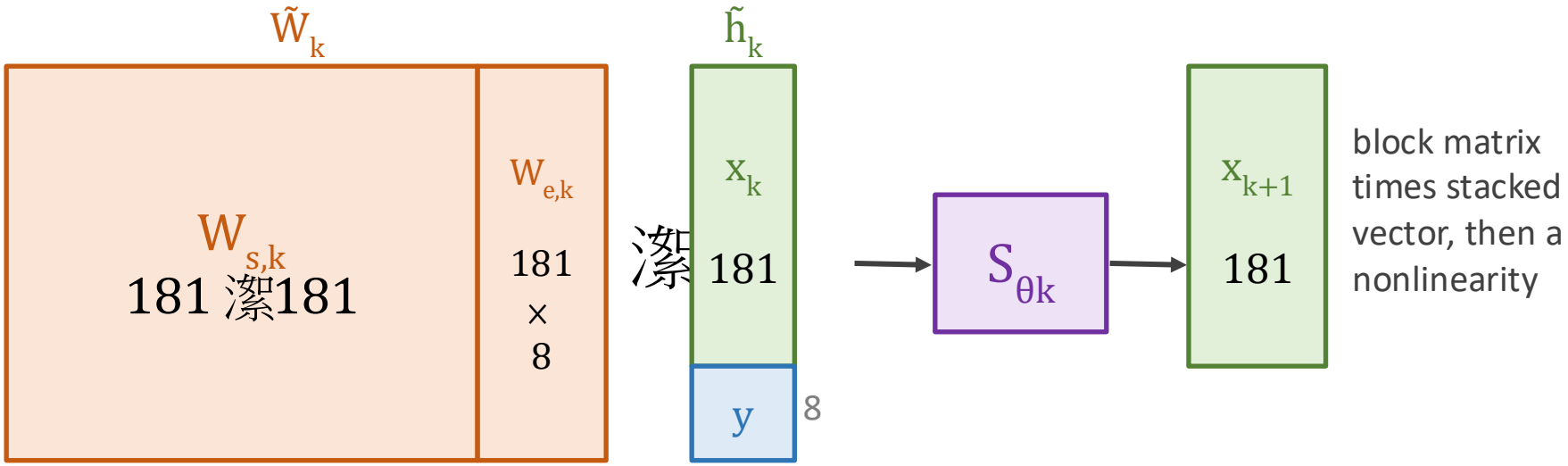
	MLP	LISTA
intermediate variable	h_k	x_k (a source map)
trainable transformation	W_k, b_k	$W_{s,k}$, $W_{e,k}$, θ_k
nonlinearity	σ (ReLU, …)	$S_{\theta k}$ (soft threshold)
trained by	backpropagation	backpropagation

$W_{e,k}y$ sits where the bias b_k sits, but it is not fixed: it changes with every measurement y, which enters every layer.

Familiar neural-network operations, arranged by an optimization algorithm.

The bridge: a LISTA layer is a feedforward layer

$$\tilde{h}_k = [\, x_k \,;\, y \,] \qquad \tilde{W}_k = [\, W_{s,k} \;\; W_{e,k} \,]$$



(sizes for our array; y block not to scale)

$$x_{k+1} = S_{\theta k}(\tilde{W}_k \, \tilde{h}_k)$$

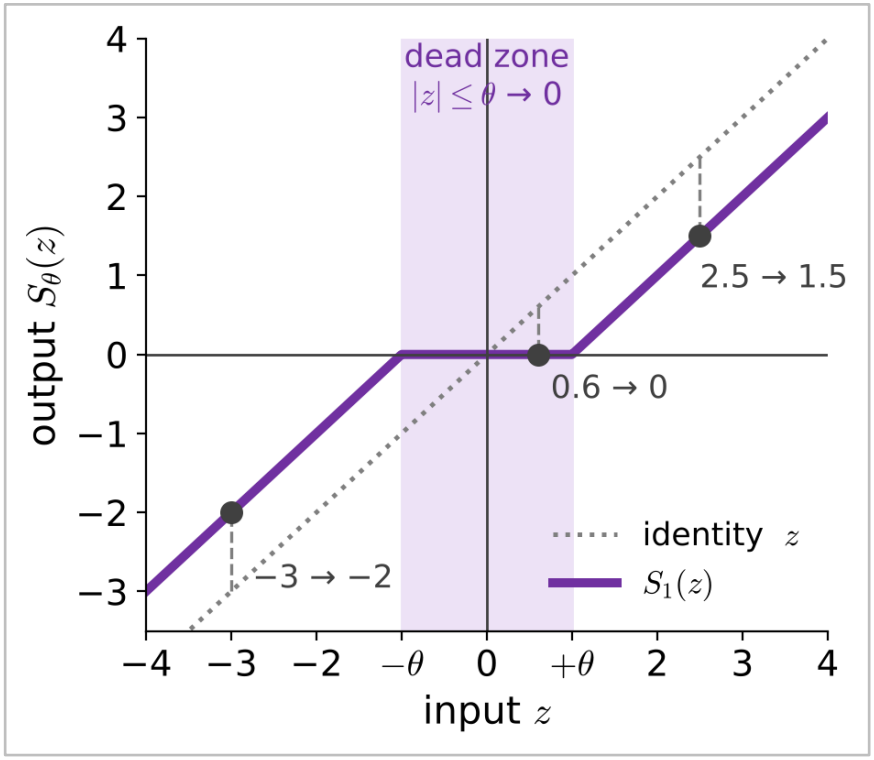
LISTA layer

$$h_{k+1} = \sigma(W_k \, h_k + b_k)$$

MLP layer

Q2 Why are $W_{s,k}$ and $W_{e,k}$ ordinary weight matrices, and not the outputs of another network?

From proximal operator to neural-network activation



Small coefficients → exactly zero
Large coefficients → shrunk toward zero by θ

$$S_1(0.6) = 0 \qquad S_1(2.5) = 1.5 \qquad S_1(-3) = -2$$

LISTA LAYER

$$x_{k+1} = S_{\theta k}(W_{s,k}\, x_k + W_{e,k}\, y)$$

ORDINARY NEURAL-NETWORK LAYER

$$a_{k+1} = \varphi(W_k\, a_k + b_k)$$

- $W_{s,k}$, $W_{e,k}$ ☐ W_k: trainable linear maps
- S ☐ ϕ: activation, here derived from the L1 prior
- θ_k: learnable threshold (ISTA fixes θ = ηλ)

OTHER PRIORS, OTHER DERIVED OPERATORS

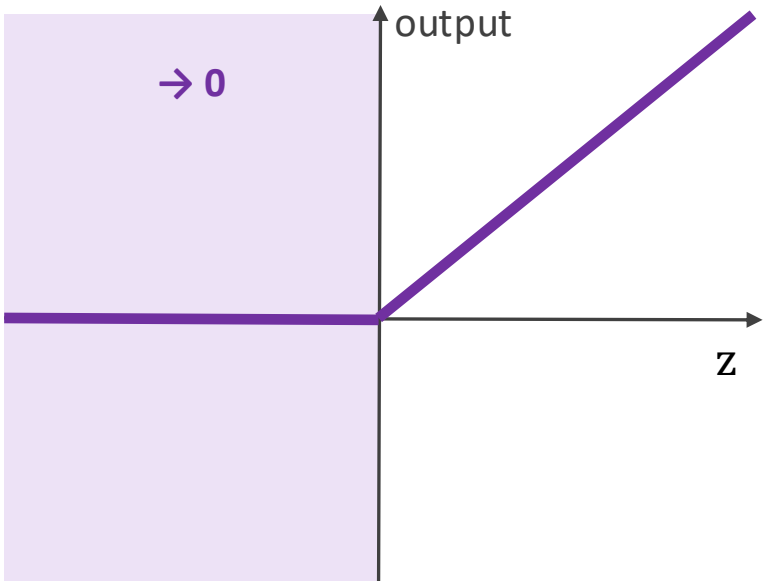
Prior or constraint	Proximal operation
L1 sparsity $\theta\|x\|_1$	Soft-thresholding $S_\theta(z)$
squared L2 $(\theta/2)\|x\|^2$	Linear shrinkage $z/(1+\theta)$
nonnegativity $x \ge 0$	ReLU $\max(0, z)$
box $a \le x \le b$	Clipping $\min(\max(z, a), b)$

Each operator is the proximal map of its prior or constraint set.

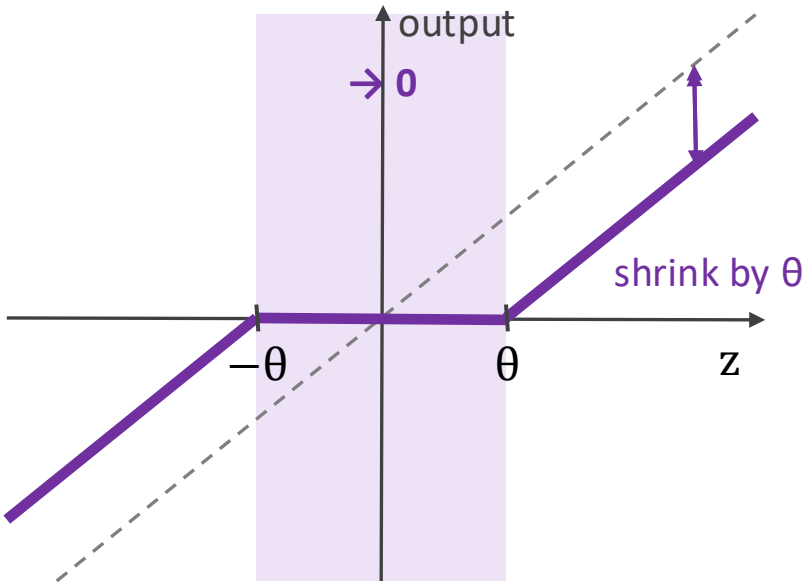
The optimization objective determines the nonlinear operator.
Learning determines how that operator is parameterized and used.

Why soft thresholding instead of ReLU?

$$\mathrm{ReLU}(z) = \max(z, 0)$$



$$S_\theta(z) = \mathrm{sign}(z) \cdot \max(|z| - \theta, 0)$$



$$S_\theta(z) = \mathrm{ReLU}(z - \theta) - \mathrm{ReLU}(-z - \theta)$$

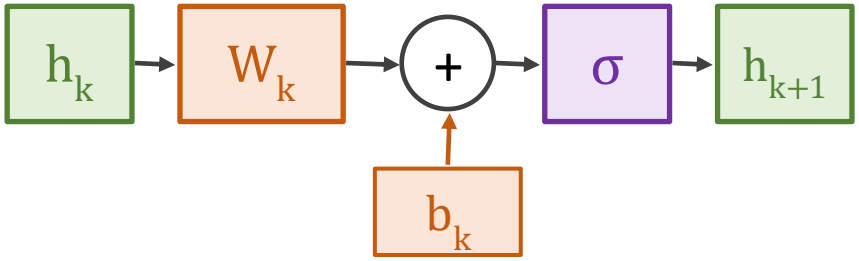
real z, θ ≥ 0

ReLU is generic. Soft thresholding is the L1 sparsity prior, in every layer.

MLP layer vs. LISTA layer: more similar than they appear

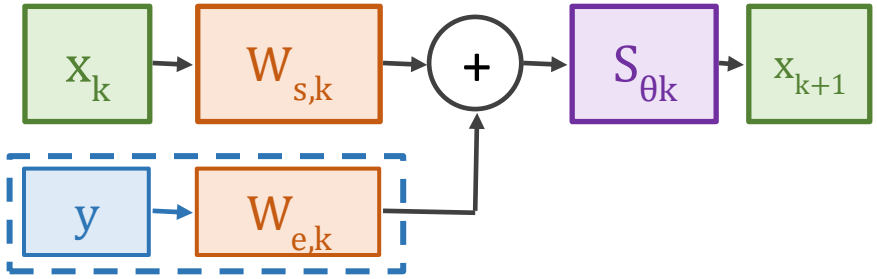
MLP layer

$$h_{k+1} = \sigma(W_k h_k + b_k)$$



LISTA layer

$$x_{k+1} = S_{\theta k}(W_{s,k} x_k + W_{e,k} y)$$



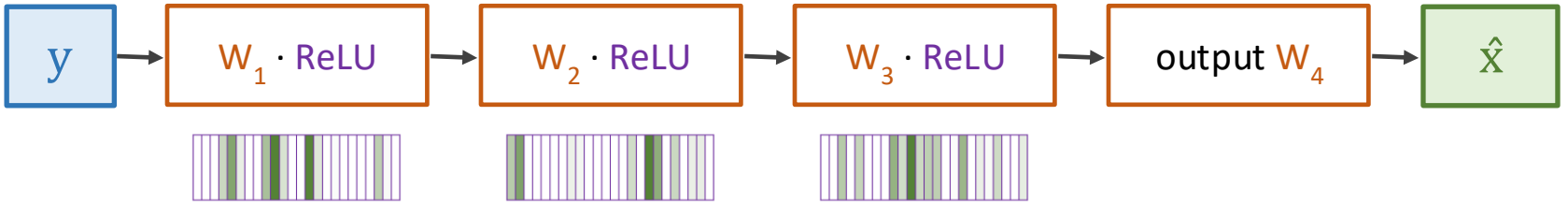
	MLP	LISTA
intermediate variable	h_k	x_k (a source map)
trainable transformation	W_k, b_k	$W_{s,k}$, $W_{e,k}$, θ_k
nonlinearity	σ (ReLU, …)	$S_{\theta k}$ (soft threshold)
trained by	backpropagation	backpropagation

$W_{e,k}y$ sits where the bias b_k sits, but it is not fixed: it changes with every measurement y, which enters every layer.

Familiar neural-network operations, arranged by an optimization algorithm.

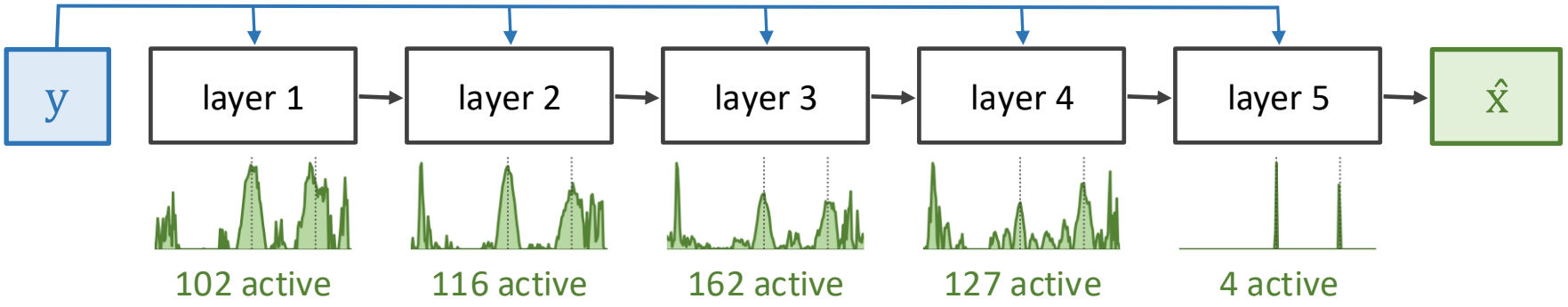
Direct MLP vs. LISTA: what happens inside?

Direct MLP: $\hat{x} = f_\phi(y)$



hidden features: generic numbers with no physical meaning (schematic)

LISTA: $x_{k+1} = S_{\theta k}(W_{s,k}x_k + W_{e,k}y)$



Real trained LISTA on a held-out scene (dotted: true speakers at −1° and +57°; each panel scaled to its own maximum).

Layers 2–4 are not clean source maps. Training only scores the final output, and layer 5 lands on −1° and +56° with 4 active angles.

LISTA's states live in source-map space, but only the final one is trained to be right.

Direct MLP vs. LISTA: what does physics contribute?

	Direct MLP	LISTA
architecture	general-purpose feedforward	unrolled sparse-recovery solver
intermediate states	generic learned features	source-map estimates x_k
nonlinearity	ReLU, GELU, …	soft thresholding $S_{\theta k}$
initialization	random	from ISTA (starts as ISTA-5)
role of physics	only through the training data	A sets the wiring and start
interpretability	limited	layers are reconstruction stages
main risk	must learn all structure from data	assumed model may not fit the scene

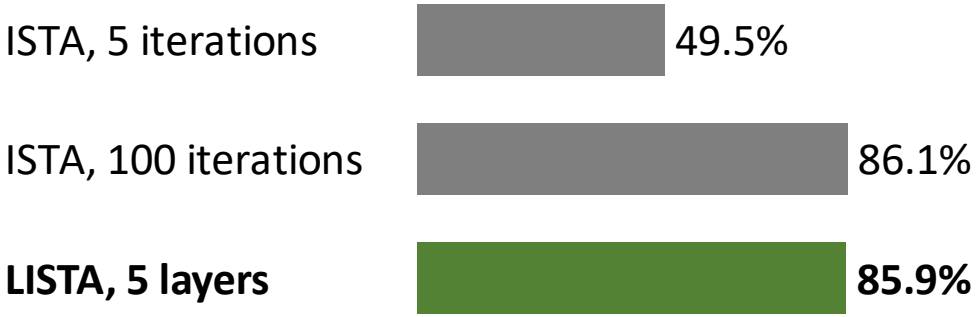
Same in both: input y, output $\hat{x}$, trained by backpropagation + an optimizer.

Q3 How is a 5-layer LISTA different from (a) 5 fixed ISTA iterations and (b) a conventional 5-layer MLP?

Same learning machinery. LISTA starts from the physics; the MLP starts from scratch.

What learning bought us here, and what it did not

Sources within ±1°, 4,000 simulated test scenes

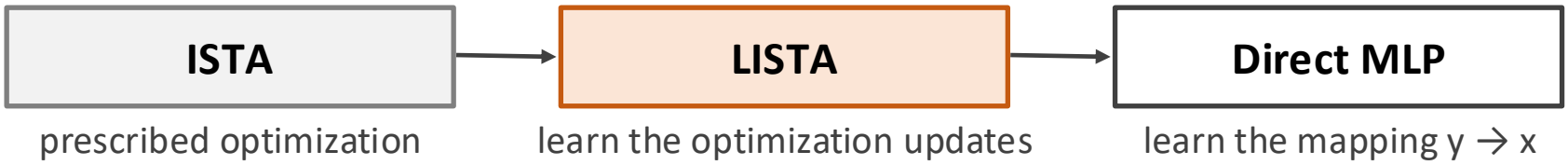


20× fewer stages, not necessarily 20× faster: each layer is a full 181 × 181 matrix.

In 4.6% of test scenes LISTA returned an empty map.

This experiment has no direct-MLP baseline.

5 learned layers ≈ 100 ISTA iterations here.

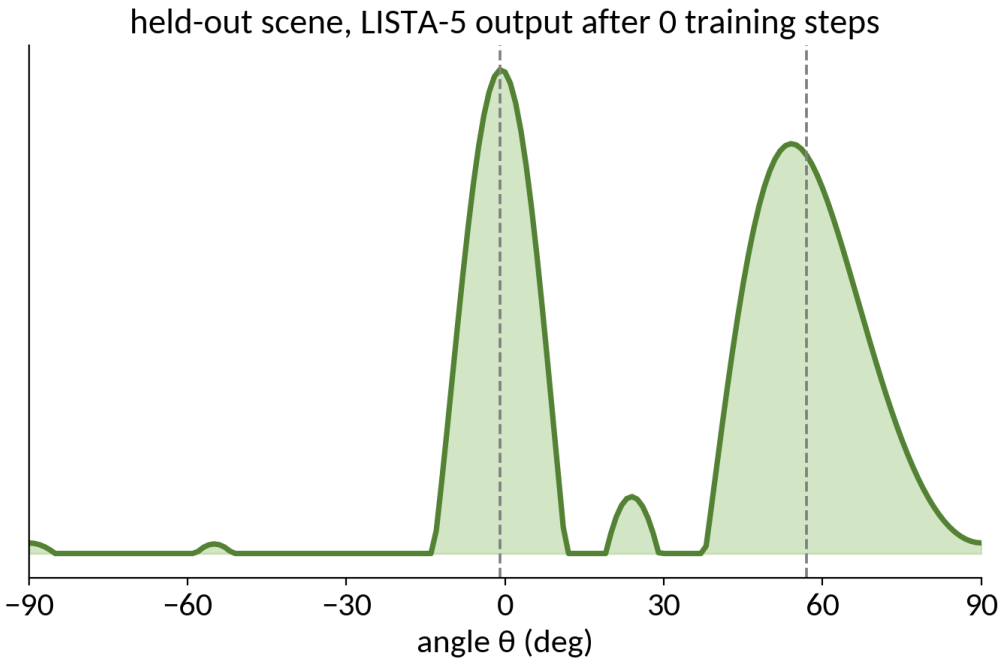


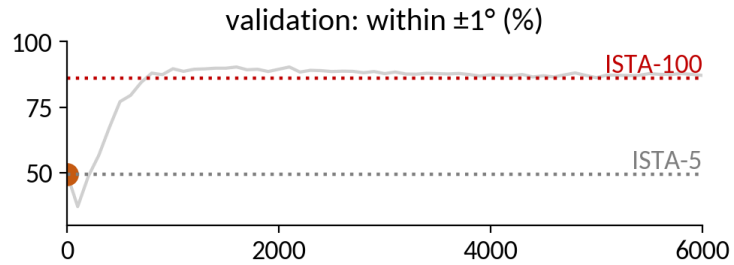
a spectrum of design choices, not a ranking

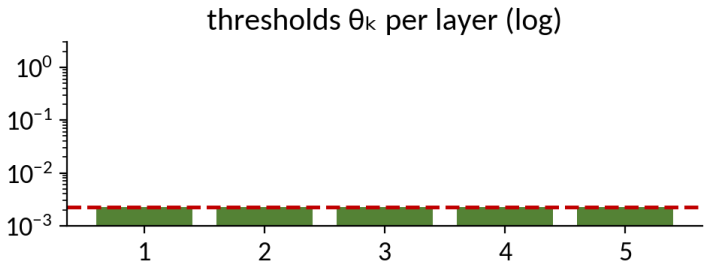
A conventional MLP learns a mapping. LISTA learns a reconstruction procedure whose architecture is inspired by physics and optimization.

The mathematics determines the structure. Backpropagation learns the parameters.

Watch LISTA learn: small-scale training run

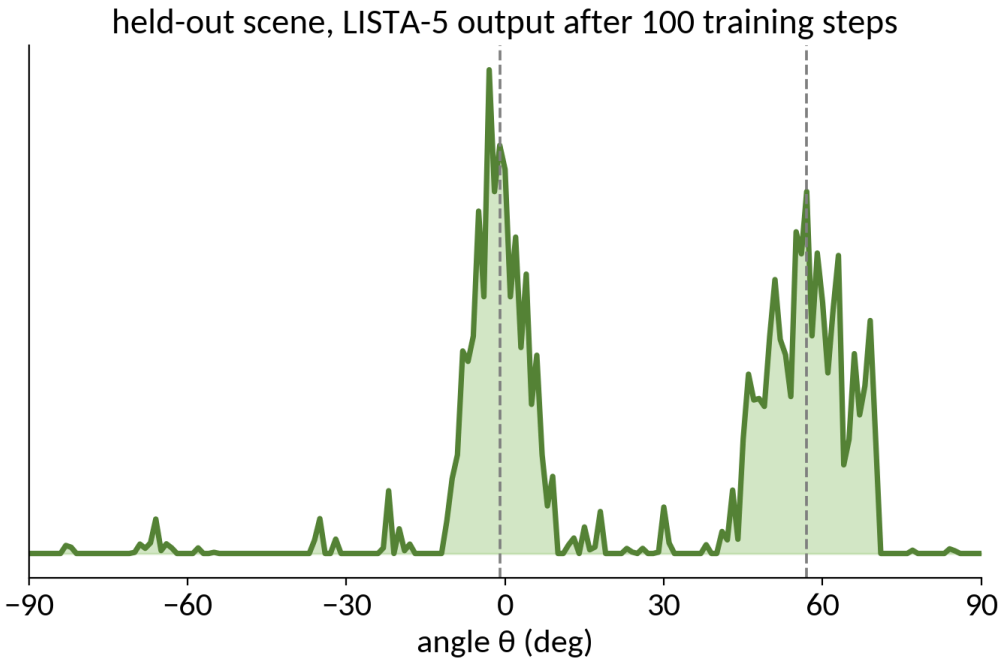


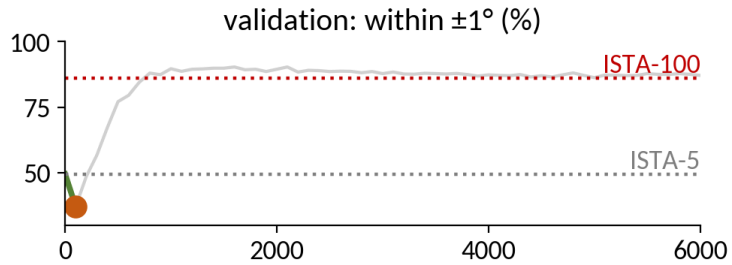


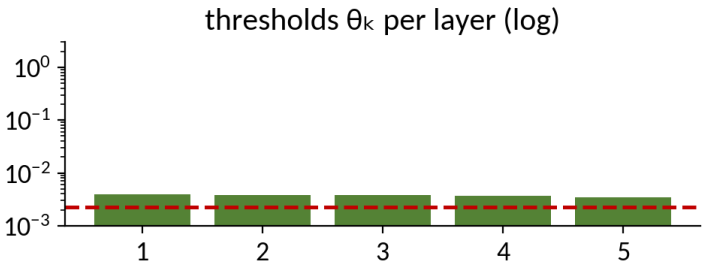


Step 0: LISTA starts out as exactly ISTA-5: broad lobes. · validation 49% within 1°

Watch LISTA learn: small-scale training run

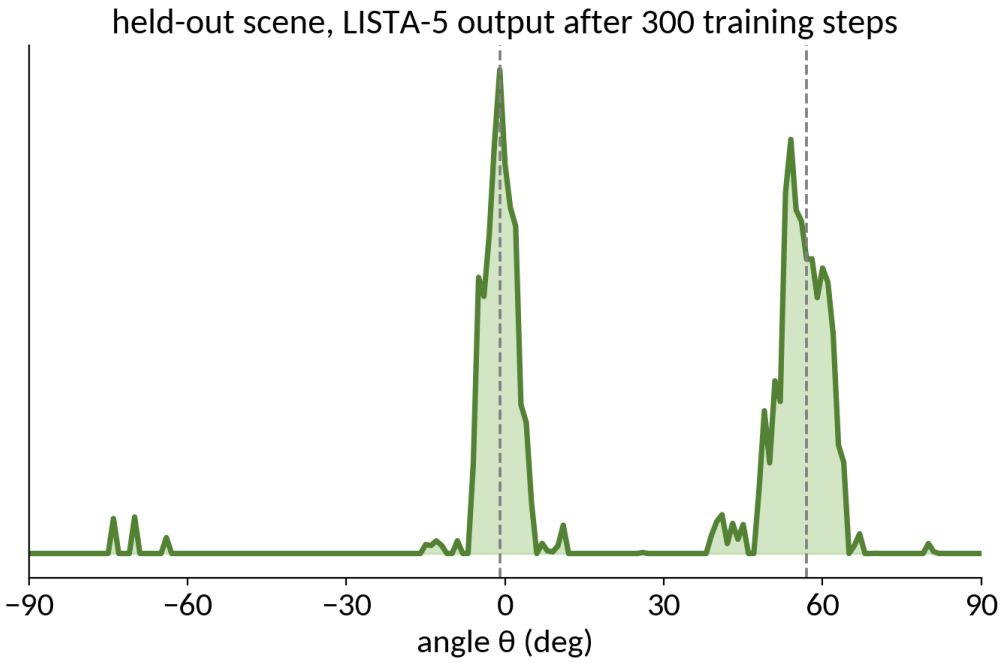


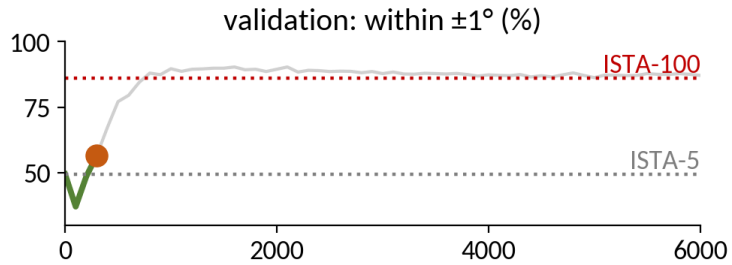


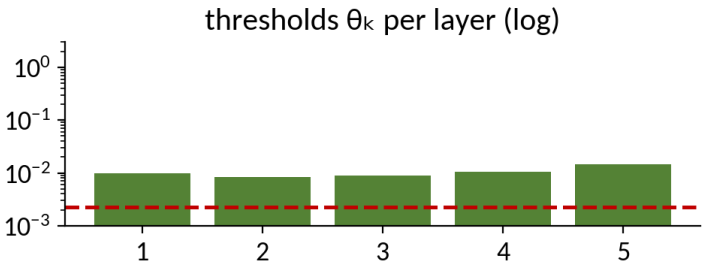


After 100 steps: accuracy first dips as parameters leave the ISTA values. · validation 37% within 出

Watch LISTA learn: small-scale training run

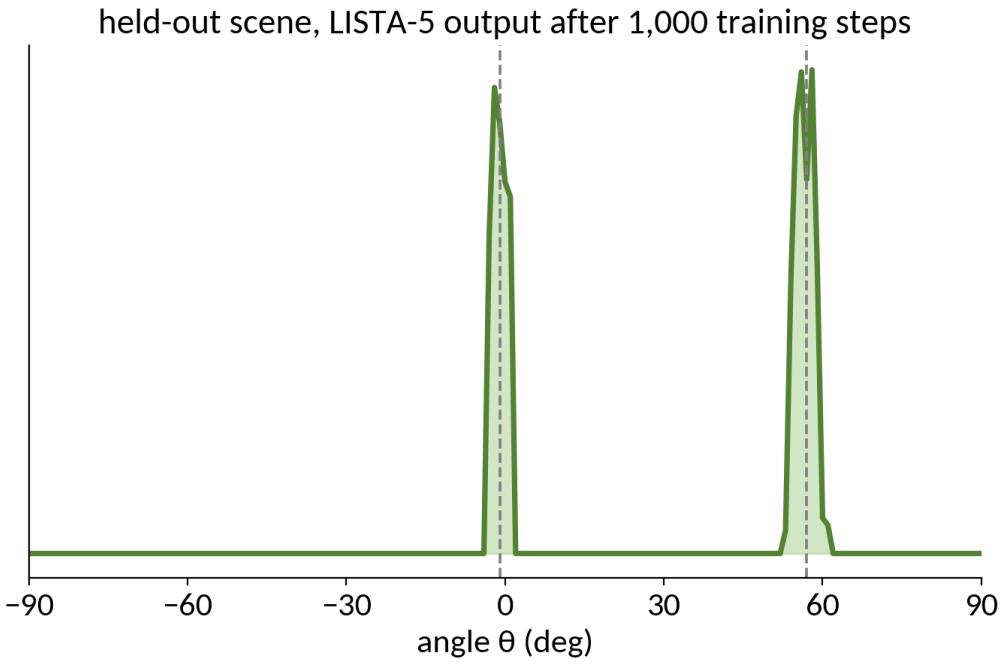


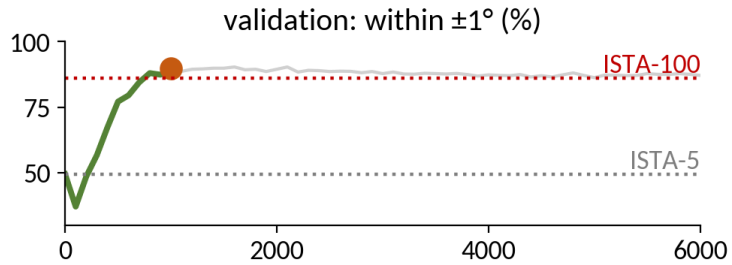


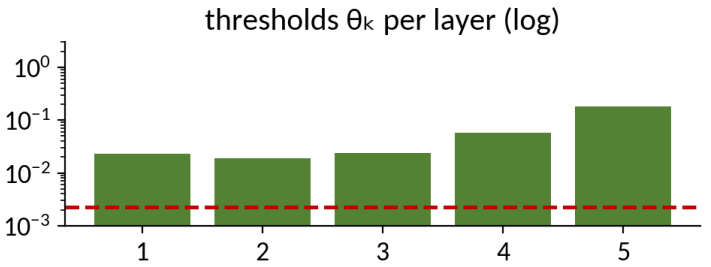


After 300 steps: recovering; the two sources begin to separate. · validation 57% within 1°

Watch LISTA learn: small-scale training run

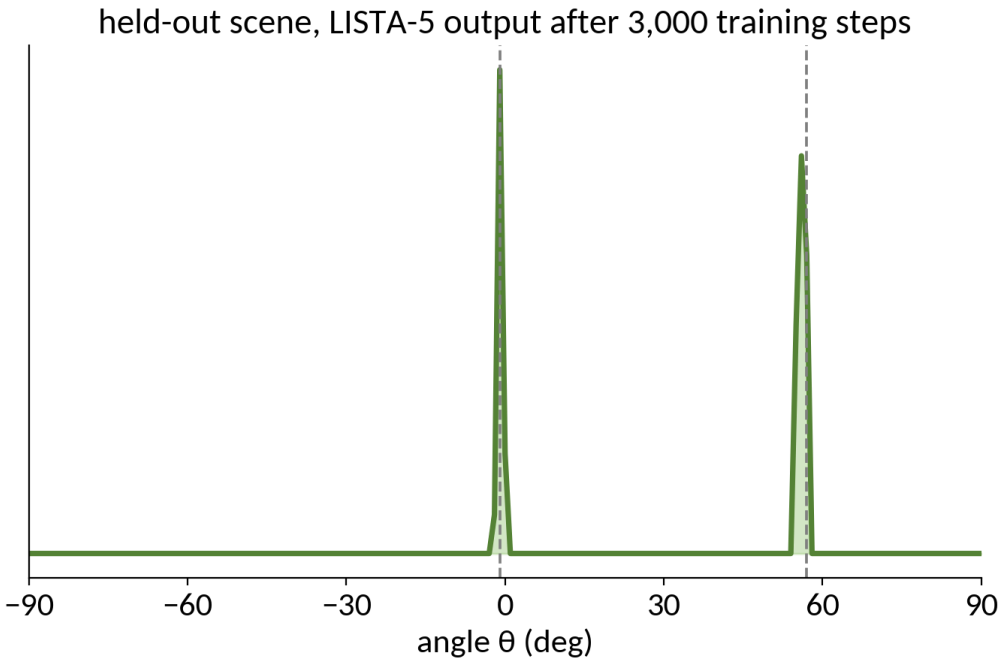


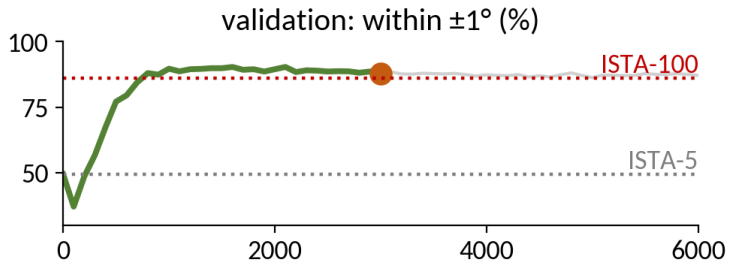


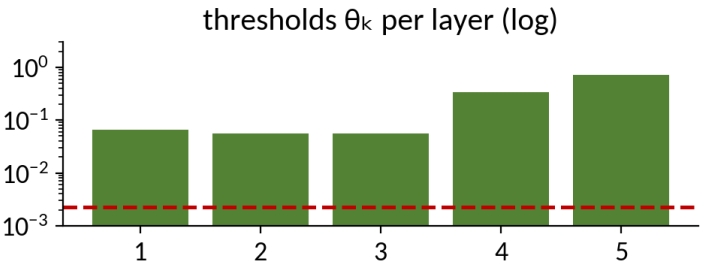


After 1,000 steps: already past ISTA-100 on validation. · validation 90% within $\pm 1^\circ$

Watch LISTA learn: small-scale training run

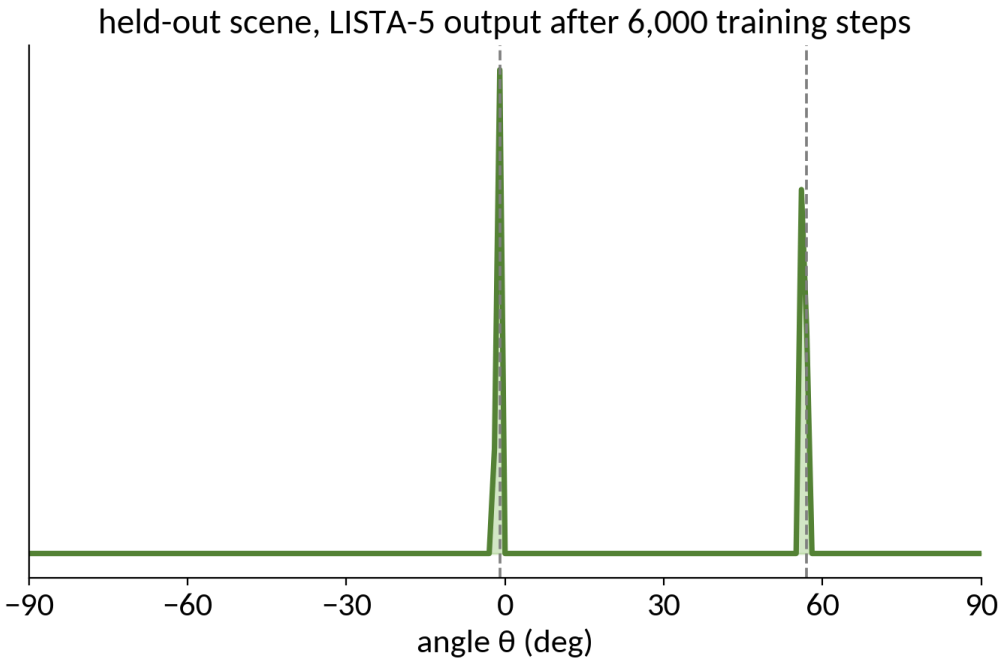


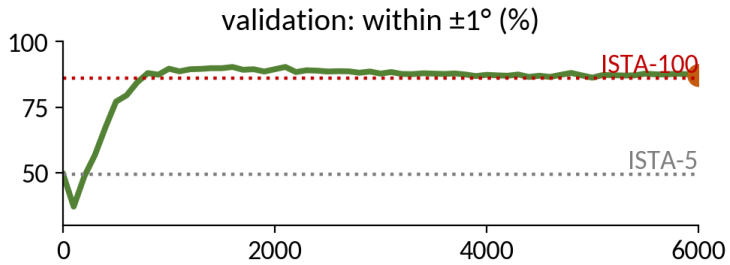


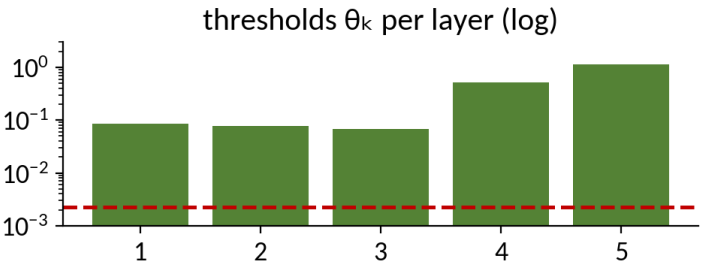


After 3,000 steps: thresholds keep growing; later layers prune hard. · validation 88% within $\pm 1^\circ$

Watch LISTA learn: small-scale training run



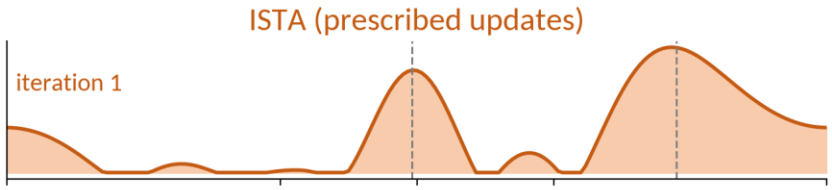


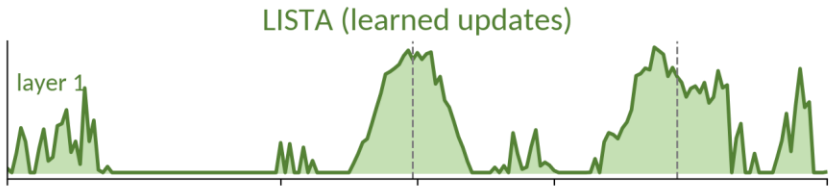


After 6,000 steps: the final network. · validation 87% within ±1°

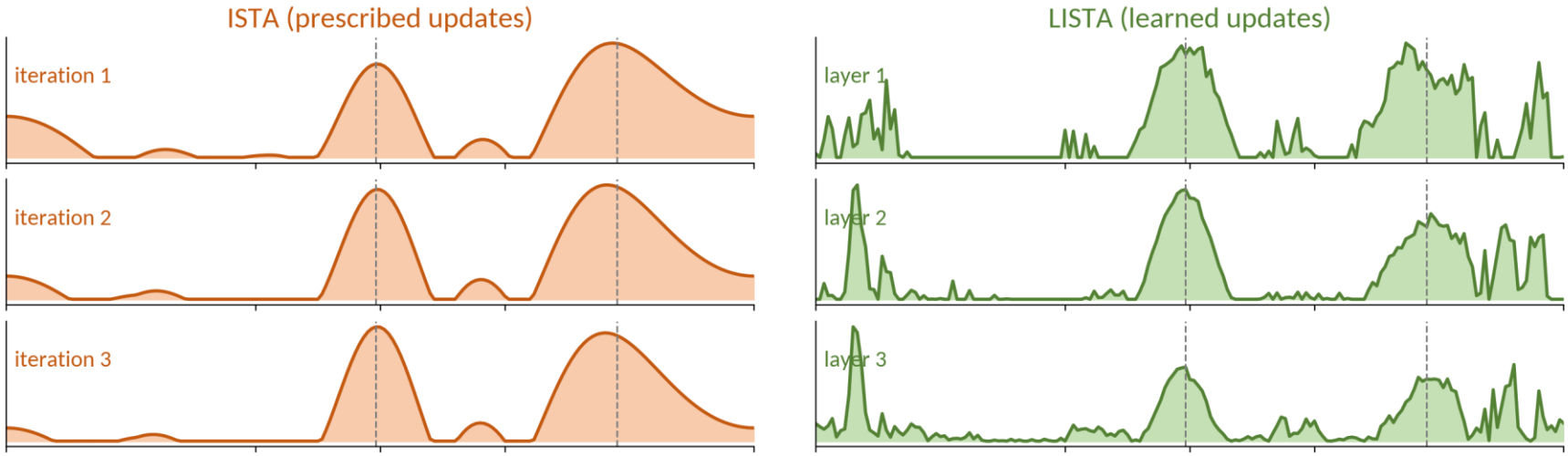
Training grew the thresholds 30–500× and sharpened five blurry steps.

Layer by layer: ISTA versus LISTA

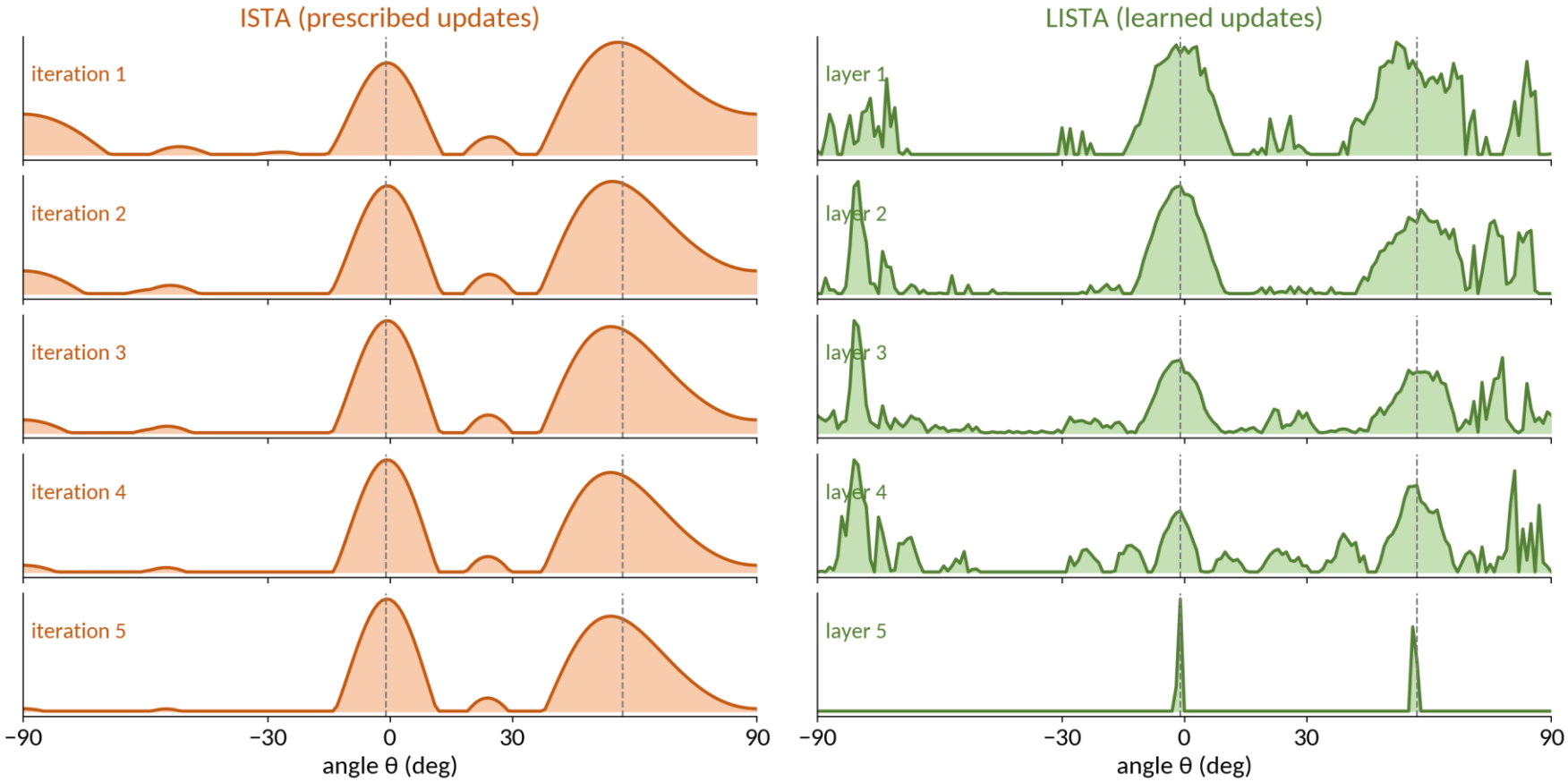




Layer by layer: ISTA versus LISTA

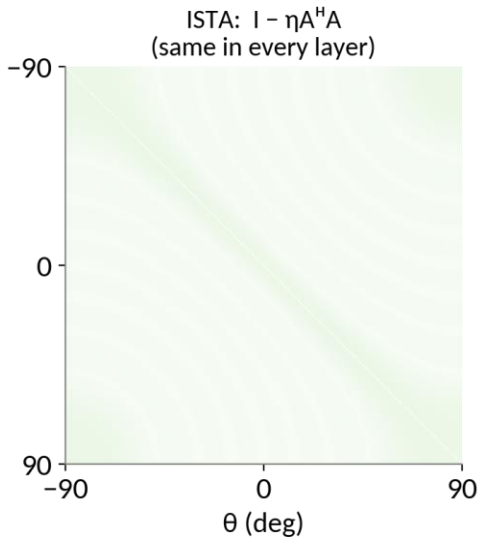


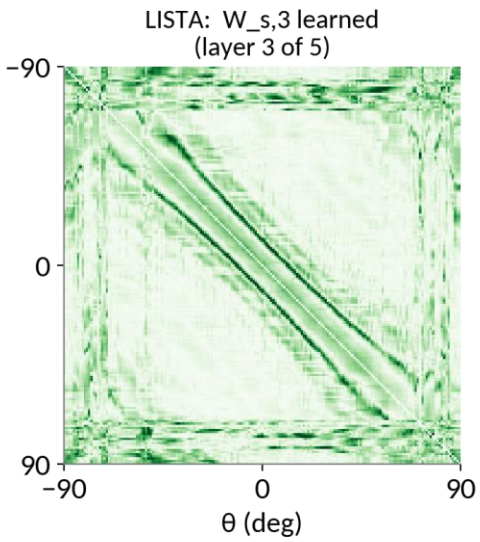
Layer by layer: ISTA versus LISTA

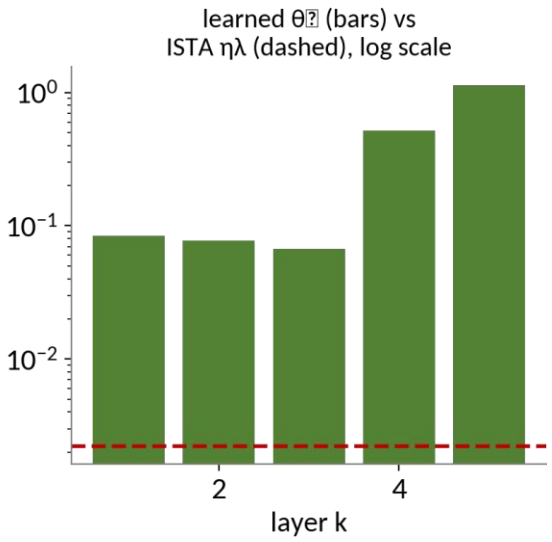


Same template, same five steps: learning changed only the numbers inside them.

What did LISTA actually learn?



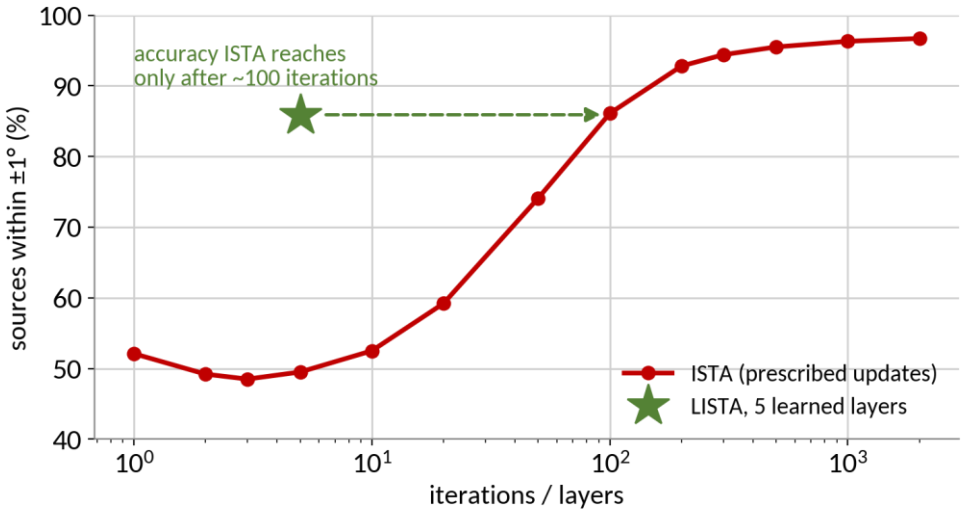




- Learned W_s keeps ISTA's band structure but reshapes it: the physics template survives training.
- Thresholds are ~30× to ~500× larger than ISTA's; the last layers prune hardest.

The learned solver is still recognizably ISTA, tuned to this array and scene class.

ISTA versus LISTA on 4,000 test scenes



Method	±1°	NMSE
ISTA-5	49.5%	−0.3 dB
ISTA-100	86.1%	−0.5 dB
ISTA-2000	96.7%	−1.4 dB
LISTA-5	85.9%	−5.0 dB

In 4.6% of scenes LISTA outputs an empty map: learned solvers can fail silently.

Five learned layers ≈ 100 ISTA iterations on localization, and the best amplitude error of any row.

Learning buys speed: ~20× fewer layers for the same localization accuracy.

The larger lesson

Hand-designed

ISTA: every parameter derived from A, η, λ. Reliable, slow.

Model-based learning

LISTA: structure fixed by physics; the numbers learned.

End-to-end

Black-box network y → x: flexible, no physics built in.

more physics built in → more learned from data

LISTA does not discard the inverse problem. It turns the structure of the solver into the architecture of the learned system.

Model-based learning sits between hand-designed and end-to-end.

Other application: Video optical flow: how did every pixel move?



I_1



I_2



$u(p)$



hue = direction
saturation = speed

Estimate a 2-D displacement at every pixel p: $u(p) = [u_x(p), u_y(p)]^T$

KNOWN STRUCTURE: BRIGHTNESS CONSTANCY

$$I_1(p) \approx I_2(p + u(p))$$

A physical point keeps its appearance at its new image location.

An approximation: it breaks at occlusions, shadows and lighting changes.

CLASSICAL OPTIMIZATION

$$\min_u \underbrace{\Sigma_p |I_1(p) - I_2(p + u(p))|^2}_{\text{match appearances}} + \lambda \underbrace{R(u)}_{\text{plausible motion}}$$

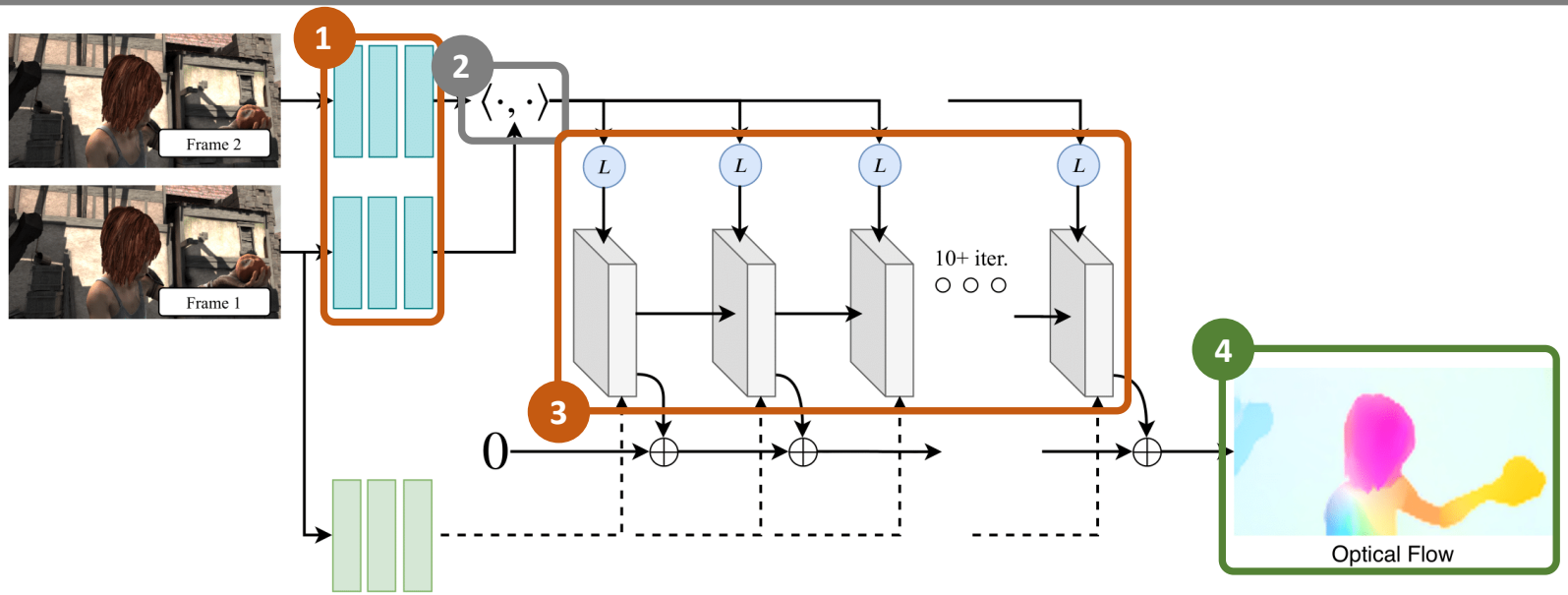
$$u_{k+1} = u_k + \Delta u_k$$

start with an estimate, find mismatches, refine

Images: Teed & Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow," ECCV 2020, Fig. 1. arXiv:2003.12039 · github.com/princeton-vl/RAFT. Color wheel: RAFT flow_viz code.

The unknown is now a motion field, but matching and refining are already known ideas.

RAFT: learning how to refine motion



壓 Learn features

$$F_1 = \varphi_\theta(I_1)$$
$$F_2 = \varphi_\theta(I_2)$$

② All-pairs correlation

$$C(p, q) = F_1(p)^T F_2(q)$$

explicit: every p vs every q

③ Recurrent update (×12)

$$u_{k+1} = u_k + \Delta u_k$$
$$\Delta u_k = G_\theta(u_k, C, h_k)$$

simplified notation; h_k = hidden state

壎 Refined flow



LISTA: learn W_s, W_e and thresholds θ_k

RAFT: learn features ϕ_θ and the motion update G_θ

RAFT is inspired by correspondence + iterative refinement; it is not an exact unrolling of a classical optimizer.

Figure: Teed & Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow," ECCV 2020, Fig. 1 (highlights added). arXiv:2003.12039

Keep correspondence and iterative refinement; learn the features and the corrections.

Questions?