Programming Languages and Analyses for Reliable, Available, and Secure Software

Michael Hicks
University of Maryland,
College Park

PL@UM
PROGRAMMING LANGUAGES UNIVERSITY OF MARYLAND

Software runs the world

Software runs the world

- We (sometimes indirectly) interact with devices running (lots of) software every day

Software runs the world

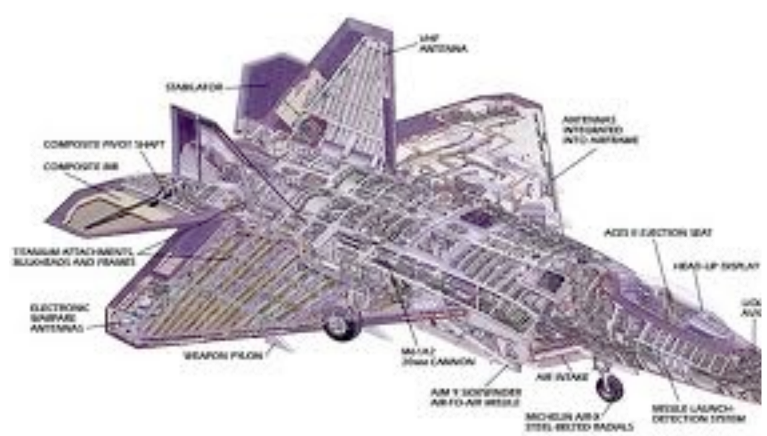
- We (sometimes indirectly) interact with devices running (lots of) software every day
  - Desktops, laptops, routers, smartphones, tablets

Software runs the world

- We (sometimes indirectly) interact with devices running (lots of) software every day
  - Desktops, laptops, routers, smartphones, tablets
  - Coffee makers, TVs, energy meters, medical devices

Software runs the world

- We (sometimes indirectly) interact with devices running (lots of) software every day
  - Desktops, laptops, routers, smartphones, tablets
  - Coffee makers, TVs, energy meters, medical devices
  - Cars, aircraft, weapon systems, nuclear centrifuges



Software **failures** are **disruptive**

Software failures are disruptive

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours

Software **failures** are **disruptive**

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem

Software failures are disruptive

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems

MIZUHO

Software **failures** are **disruptive**

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
- 3/08: Heartland exposes 134M credit cards
  - SQL injection used to install spyware

Software **failures** are **disruptive**

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
- 3/08: Heartland exposes 134M credit cards
  - SQL injection used to install spyware
- 8/07: LAX offline due to faulty network card
  - 17,000 planes grounded for eight hours

Software **failures** are **disruptive**

- 3/11: Mizuho FG's ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
- 3/08: Heartland exposes 134M credit cards
  - SQL injection used to install spyware
- 8/07: LAX offline due to faulty network card
  - 17,000 planes grounded for eight hours
- 8/03: Northeast, multi-state blackout
  - Race condition in power plant management software cascades

MIZUHO

Heartland PAYMENT SYSTEMS™
The Highest Standards | The Most Trusted Transactions

Software updates are disruptive too

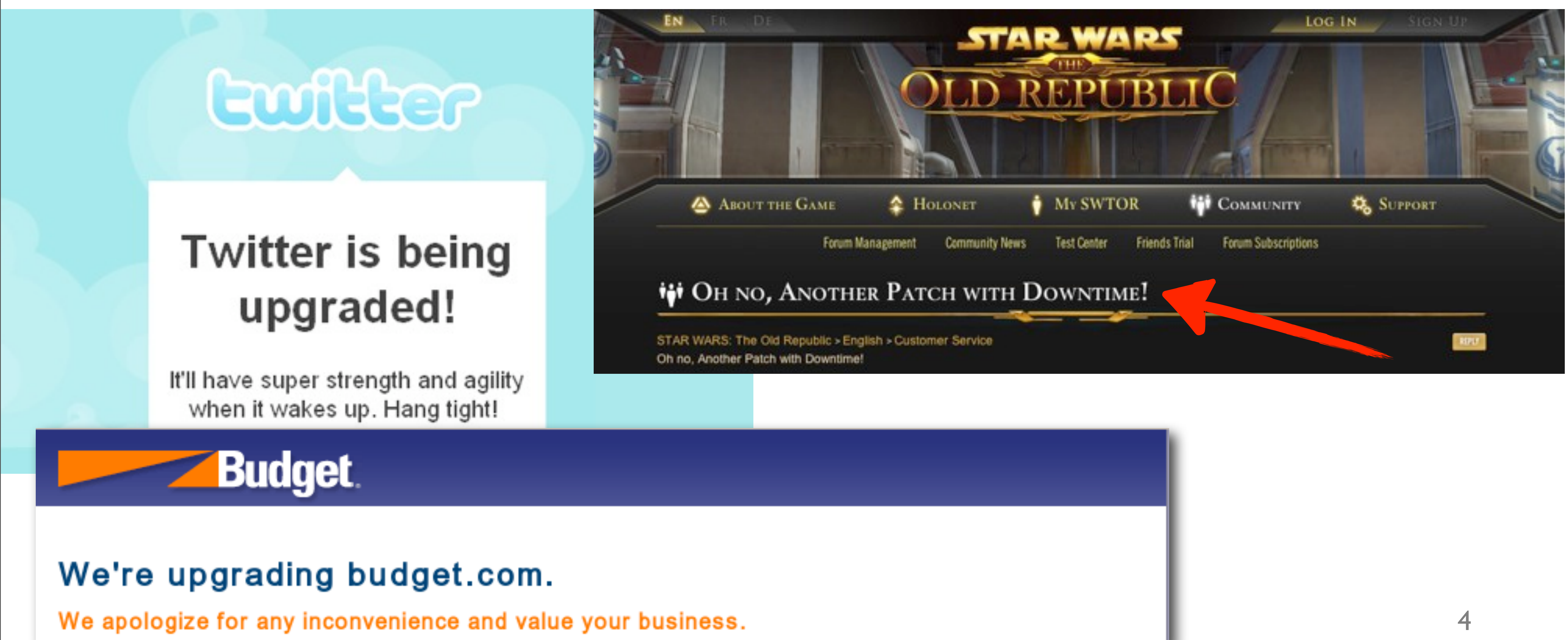
- Typically require restarting the program
  - interrupts active users / processing
  - makes services unavailable

Software updates are disruptive too

- Typically require restarting the program
  - interrupts active users / processing
  - makes services unavailable

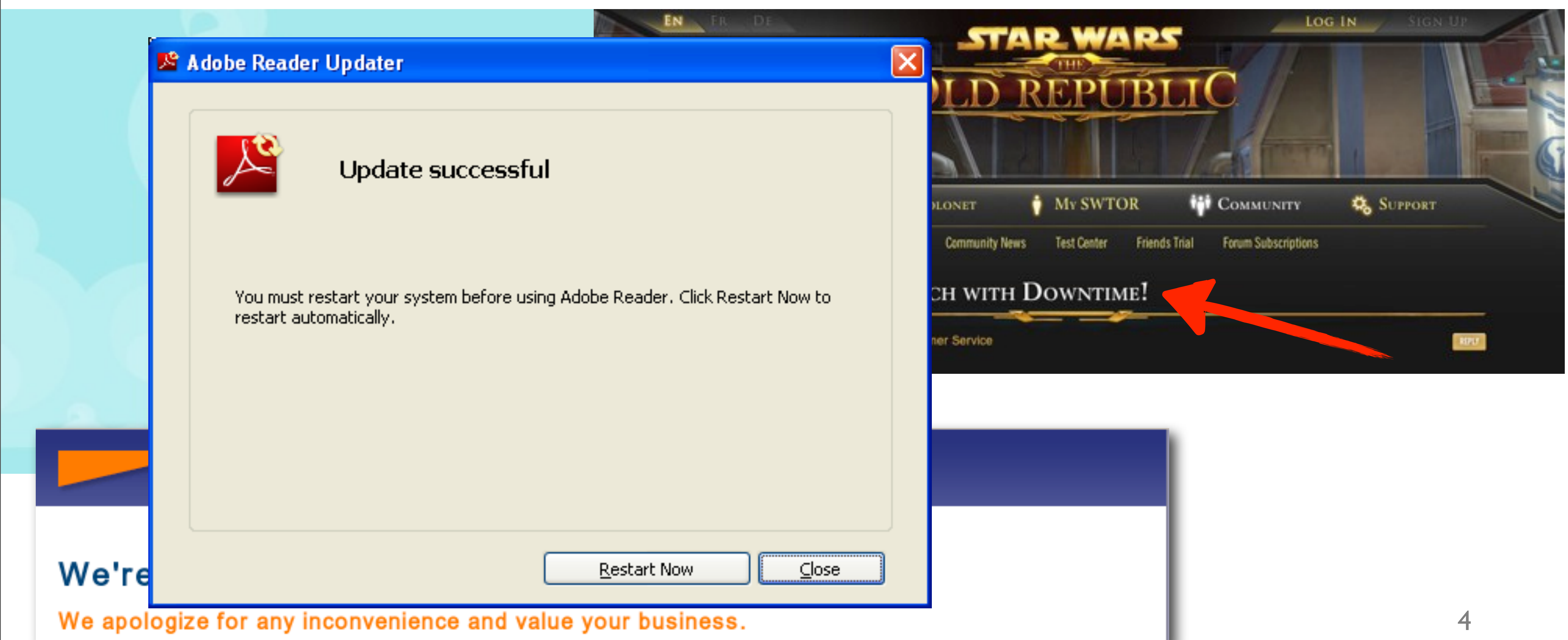
Software updates are disruptive too

- Typically require restarting the program
  - interrupts active users / processing
  - makes services unavailable



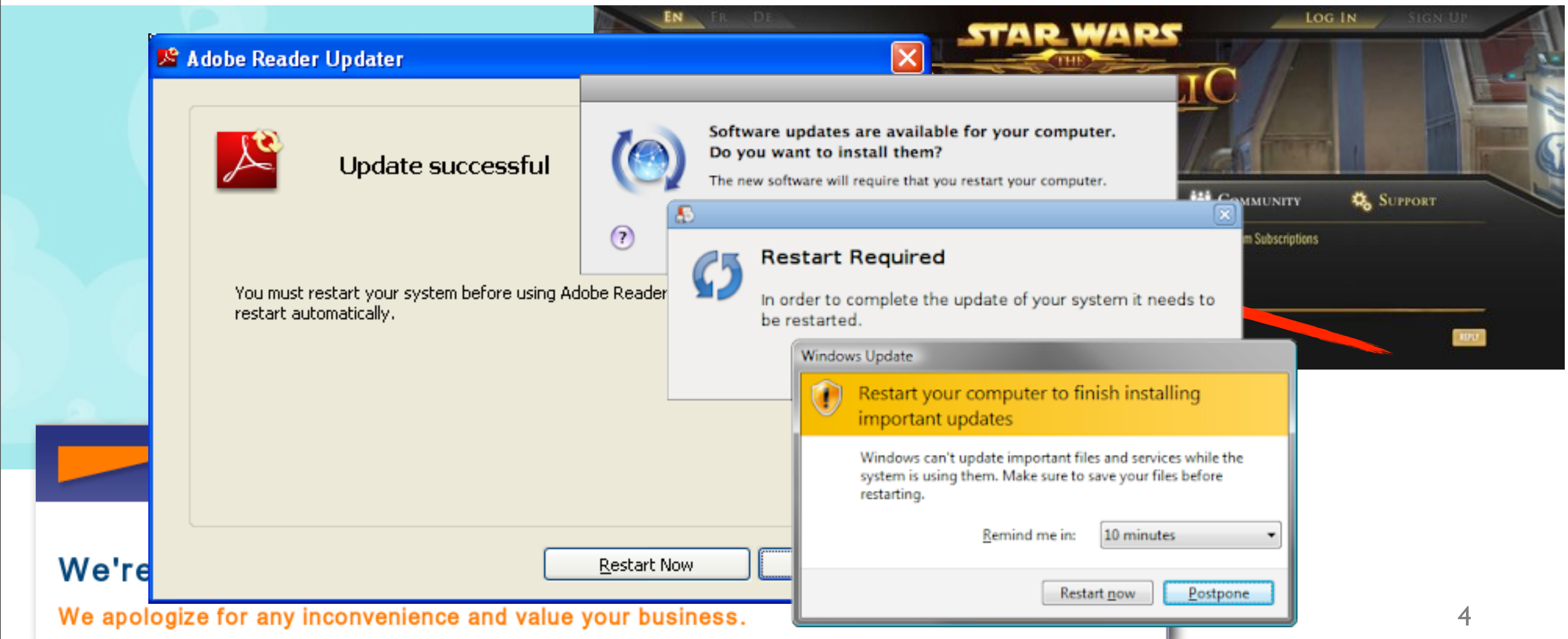
Software updates are disruptive too

- Typically require restarting the program
  - interrupts active users / processing
  - makes services unavailable



Software updates are disruptive too

- Typically require restarting the program
  - interrupts active users / processing
  - makes services unavailable



Programming Languages
A vehicle to a solution

- The language facilitates and constrains software's implementation
  - To make it easy to implement a given design
  - While discouraging/disallowing poor coding idioms
- Software tools can play a similar role
  - Enforce/encourage good coding practice
  - Simplify addition of useful features
  - Apply to existing software in existing languages

My research

- Tackles problems of software
  - reliability: software does what it should
  - security: software free from vulnerability
  - availability: avoid downtime by updating on the fly
    - and avoid delayed use of security-critical patches and upgrades

- Two-pronged approach
  - Formalize and prove key idea is correct
  - Implement and evaluate idea on real software
    - Using existing software, or write new software in new language

Roadmap

- Dynamic software updating (DSU)
  - Kitsune: Flexible and Efficient DSU for C programs
- Program analysis for security and reliability
  - Knowledge-based security: quantitatively tracking information
- Quick tour of some other work

Dynamic Software Updating (DSU)

Dynamic Software Updating (DSU)

- Goal: **Update programs while they run**
  - Avoid interruptions
    - Overwhelming number of security breaches due to unpatched software
  - Preserve critical program state

Dynamic Software Updating (DSU)

- Goal: **Update programs while they run**
  - Avoid interruptions
    - Overwhelming number of security breaches due to unpatched software
  - Preserve critical program state
- Useful for:
  - Non-stop services
    - E.g., Financial processing, air traffic control, network infrastructure
  - Programs with long-lived connections
    - E.g., OpenSSH and media streaming
  - Long-running programs with large in-memory state
    - E.g., operating systems, caching servers, in-memory databases

Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...

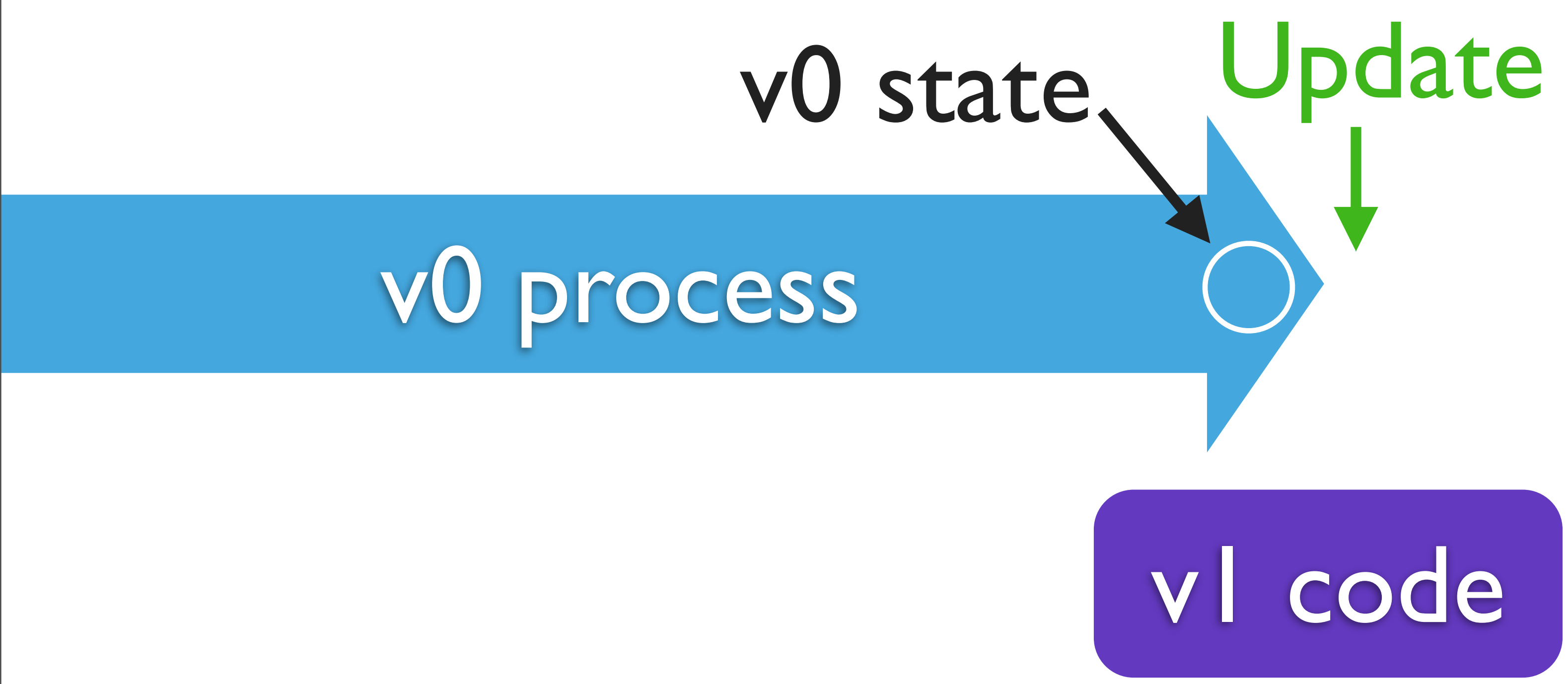
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...

Dynamic Software Updating (DSU)

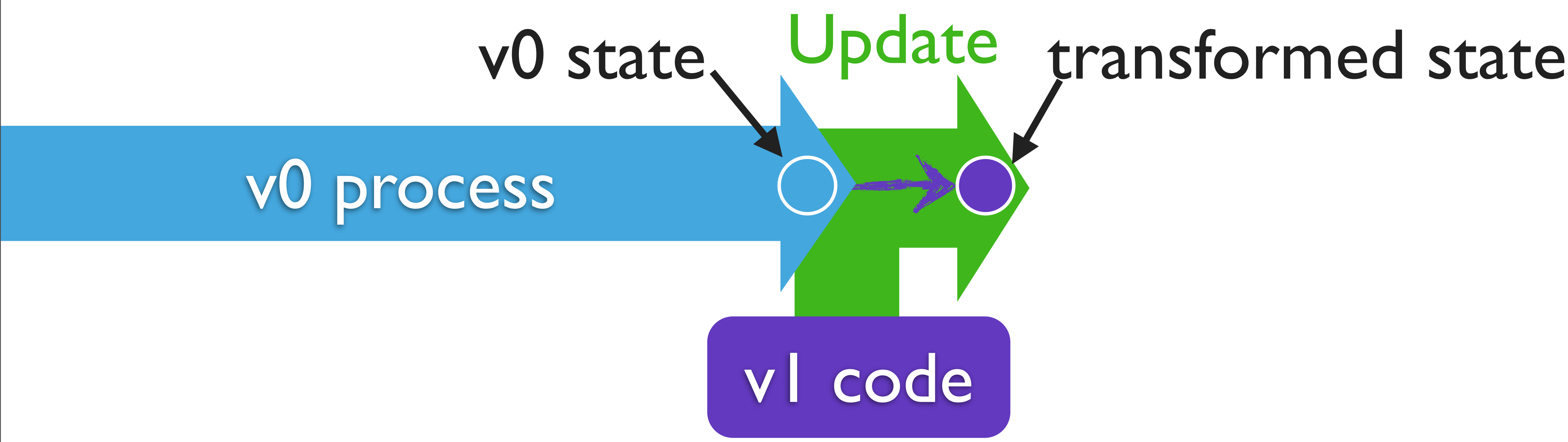
- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...

Dynamic Software Updating (DSU)



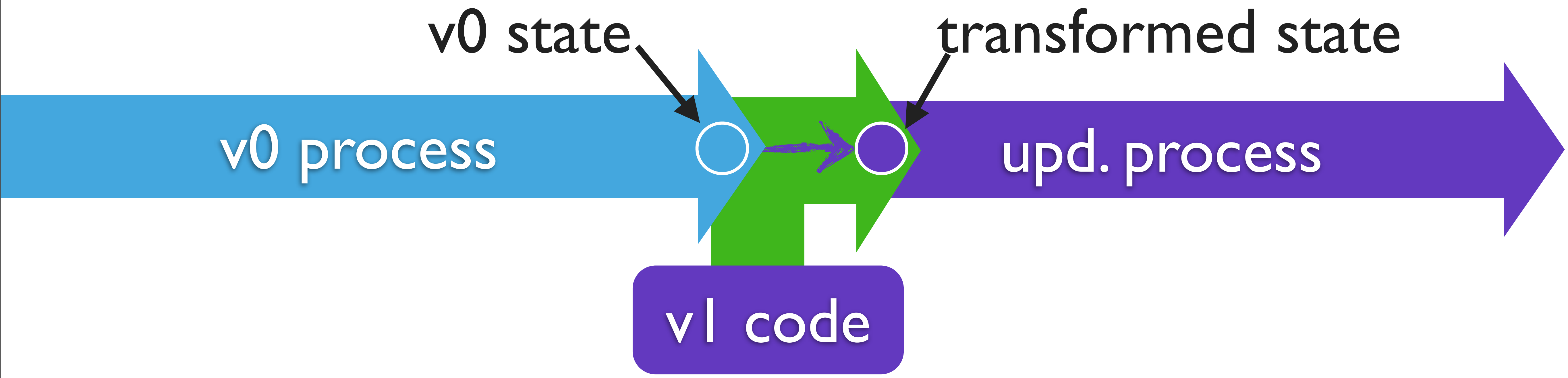
- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
    - existing connections, important data on the stack and heap, program counter, ...

Dynamic Software Updating (DSU)



- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...

Dynamic Software Updating (DSU)



- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...

Many forms of DSU now mainstream

Many forms of DSU now mainstream

language run-times

Many forms of DSU now mainstream

language run-times

app. tools

Many forms of DSU now mainstream

small TALK

JAVA™

Microsoft® .NET

APPLICATION ENHANCER 2.5

LiveRebel

Ksplice®

Bought by Oracle in 2011

language run-times

app. tools

OSes

DSU research challenges

- Which mechanisms should we use to update a running program/service?
  - Compilers, binary rewriters, run-time systems, VMs, process migration, ...
- How do we ensure a dynamic update is correct?
  - Formal specifications, static analyses, testing tools, ...
- How do we balance various competing concerns?
  - Flexibility, efficiency, ease-of-use, portability, ...

Our research in DSU

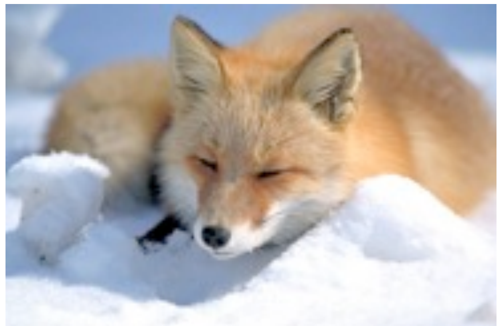
- We have thoroughly researched these questions
  - We have built DSU implementations for C and Java [PLDI'06, PLDI'09x2, HotSWUp'10, OOPSLA'12]
  - We have experience performing dozens of real-world updates on a wide variety of programs
  - We have developed methods for systematic testing and static analysis to reason about dynamic updates [POPL'05, TOPLAS'07, POPL'08, HotSWUp'10, VSTTE'12]
  - We have developed and empirically validated a variety of automatic safety checks for ensuring safety [TSE'11]
- Next: Kitsune, new DSU system for C [OOPSLA'12]

DSU state of the art: Transparency

- Goal: work on any program, with no changes
- Assessment: Laudable, but highly impractical
  - At odds with the reasons people use C
    - Control over low-level data representations, explicit resource management, legacy code, high performance
  - Empirical study shows existing transparent update approaches allow incorrect updates [TSE'11]
  - Not as transparent as they seem
    - Often requires refactoring to permit future updates
    - and/or requires satisfying a conservative static pointer analysis

New approach: Kitsune

- Favors **explicitness** over **transparency**
  - Kitsune treats DSU is a **program feature** and helps developers implement and maintain it as such
- Having the developer orchestrate DSU allows:
  - simpler DSU mechanisms
  - easier developer reasoning
  - full flexibility
  - better performance and control
- Principle: Pay for what you use
  - Design carefully builds on lessons from earlier work



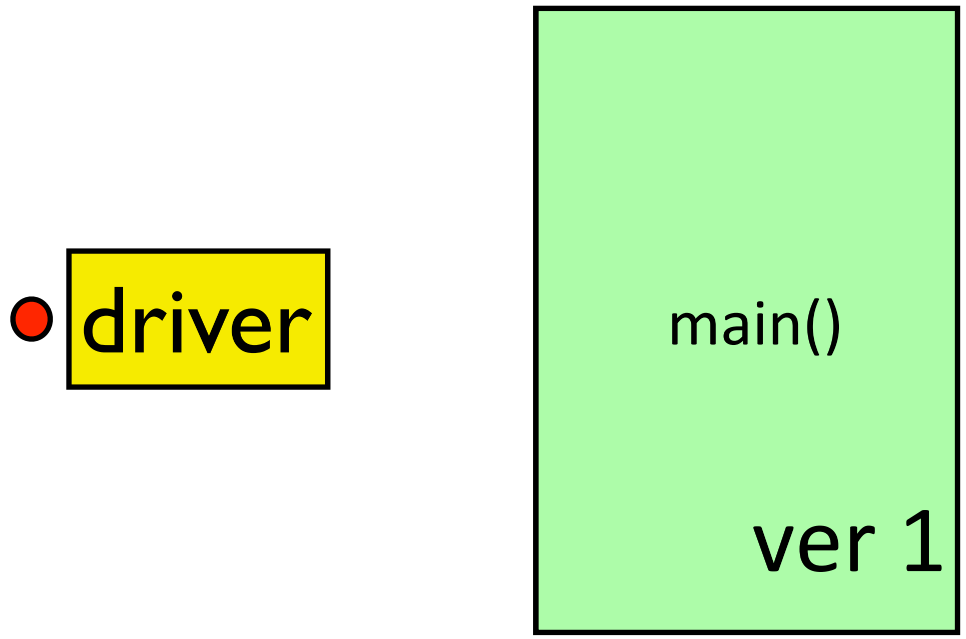
***Kitsune** (fox) - a shapeshifter according to Japanese folklore*

Results

- Applied Kitsune to six open-source programs
  - memcached, redis, icecast, snort: 3-6 mos. of releases
  - Tor, vsftpd: 2, and 4, years of releases, respectively
- Performance overhead in the noise
- Update times typically less than 40ms
- Programmer effort manageable
  - 50-160 LOC per program (largely one-time effort)
    - Program sizes from 5KLOC up to 220KLOC
  - 27-200 LOC of xfgen specs across *all* releases
    - xfgen is our DSL for writing state transformer functions

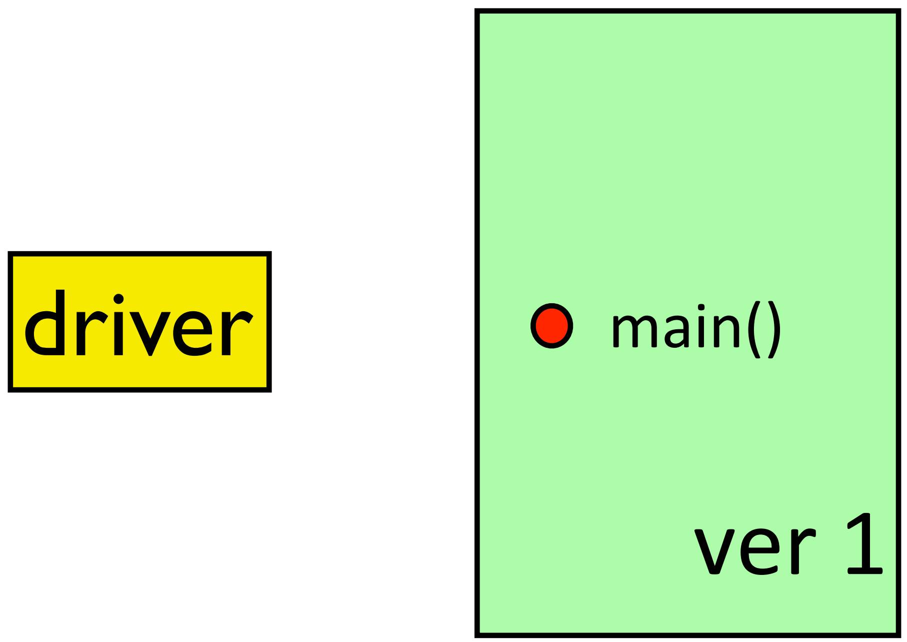
Kitsune: whole-program updates

Kitsune: whole-program updates



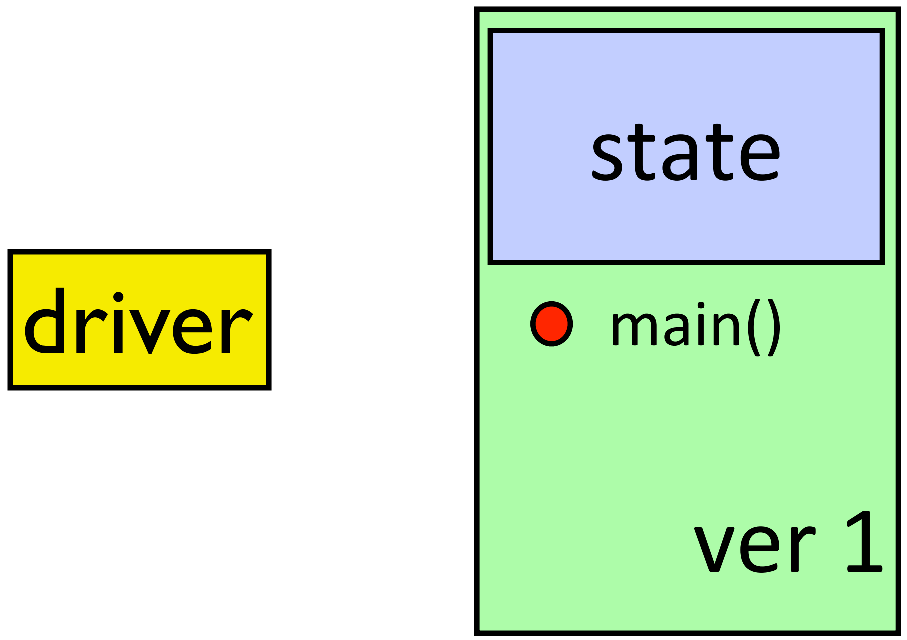
1. Load first version

Kitsune: whole-program updates



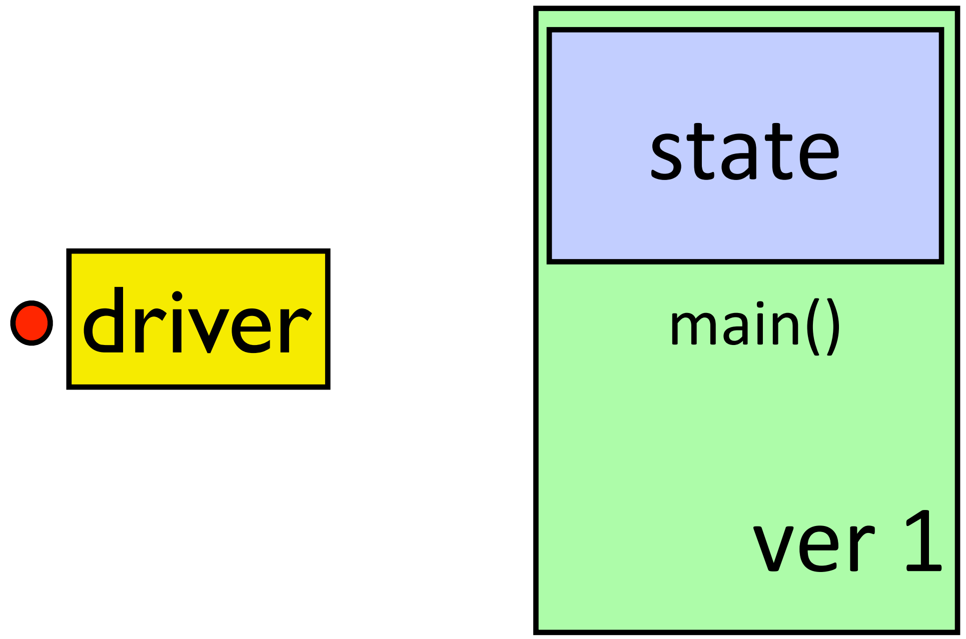
1. Load first version
2. Run it

Kitsune: whole-program updates



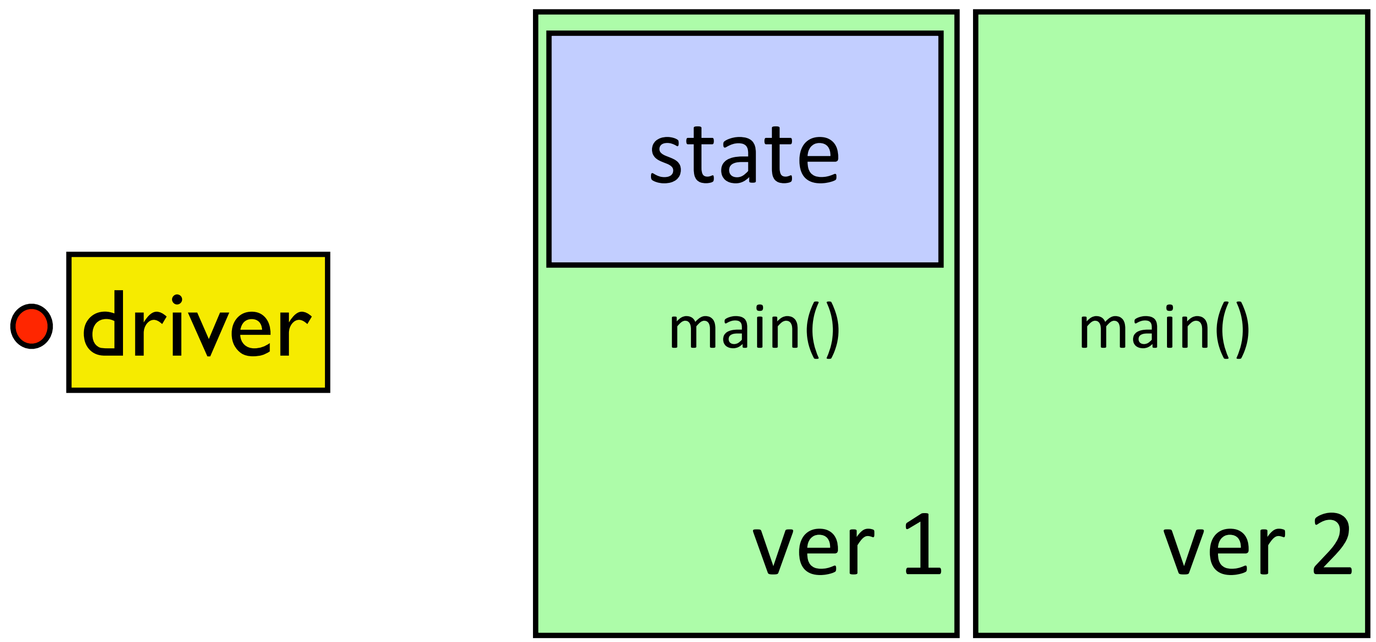
1. Load first version
2. Run it

Kitsune: whole-program updates



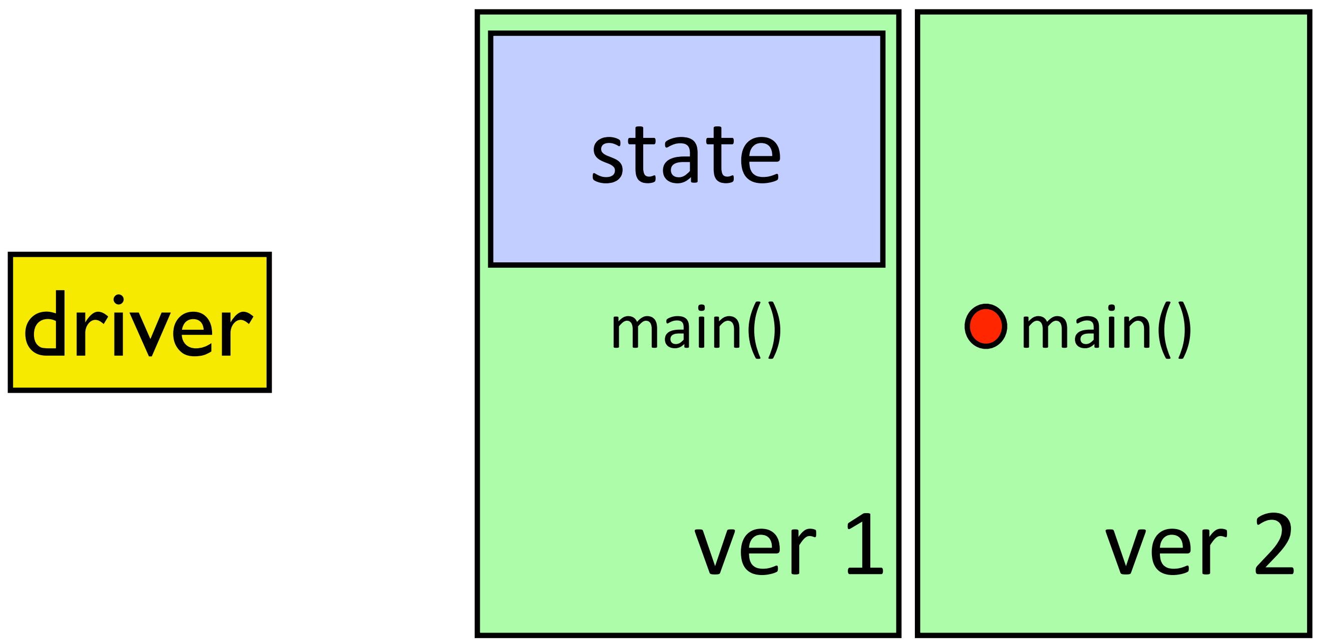
1. Load first version
2. Run it
3. Call back to driver when update ready

Kitsune: whole-program updates



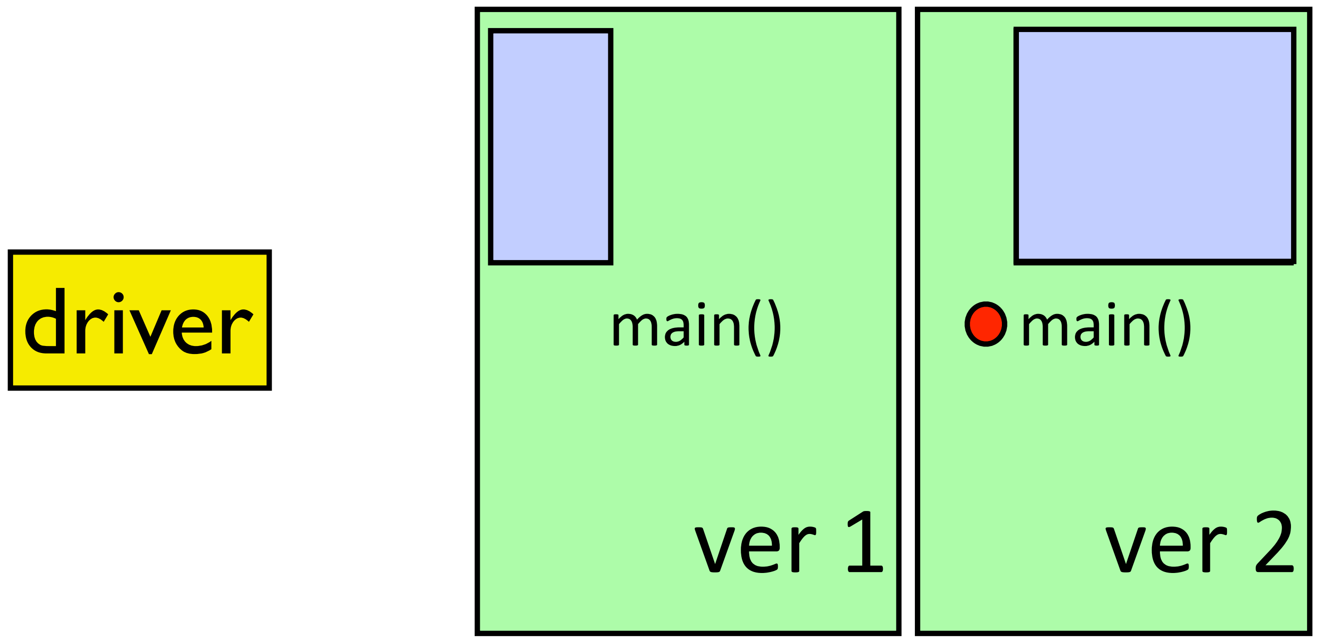
1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version

Kitsune: whole-program updates



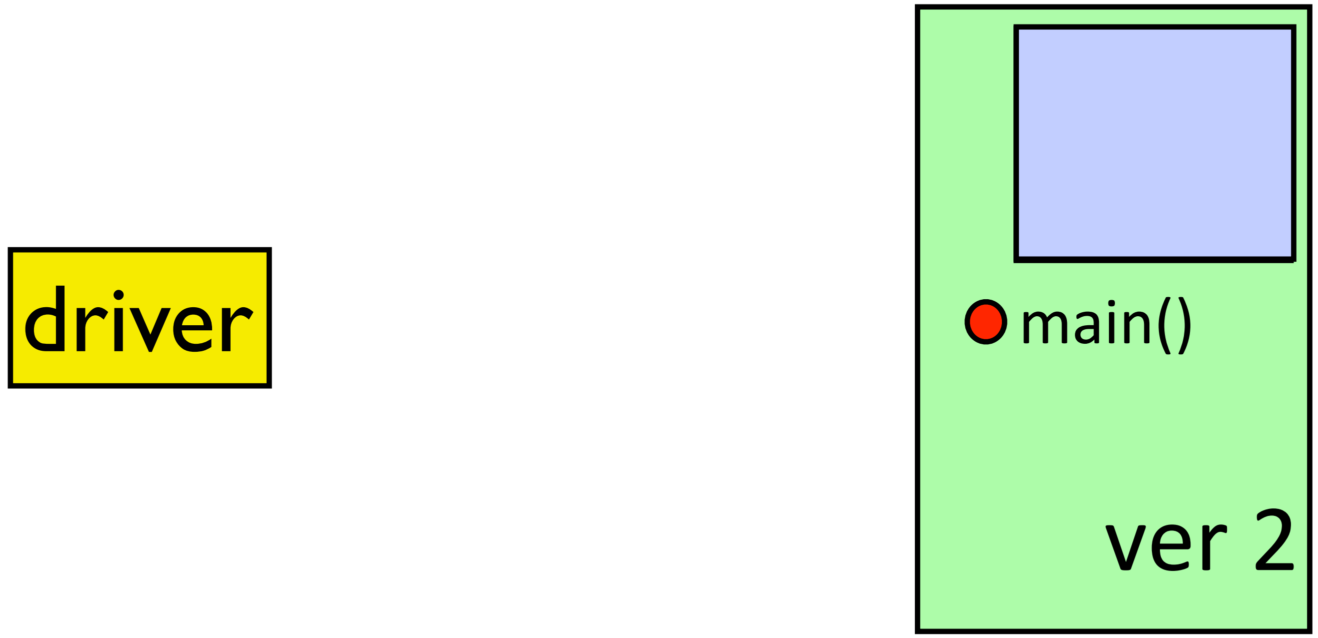
1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version

Kitsune: whole-program updates



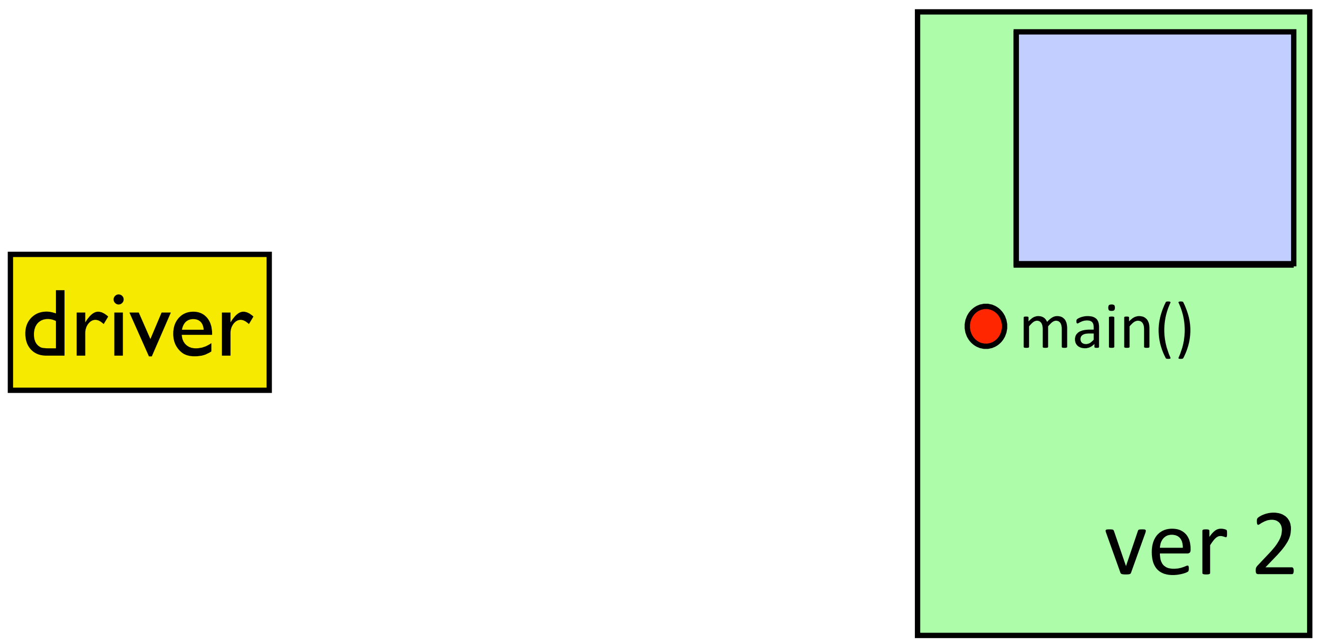
1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state

Kitsune: whole-program updates



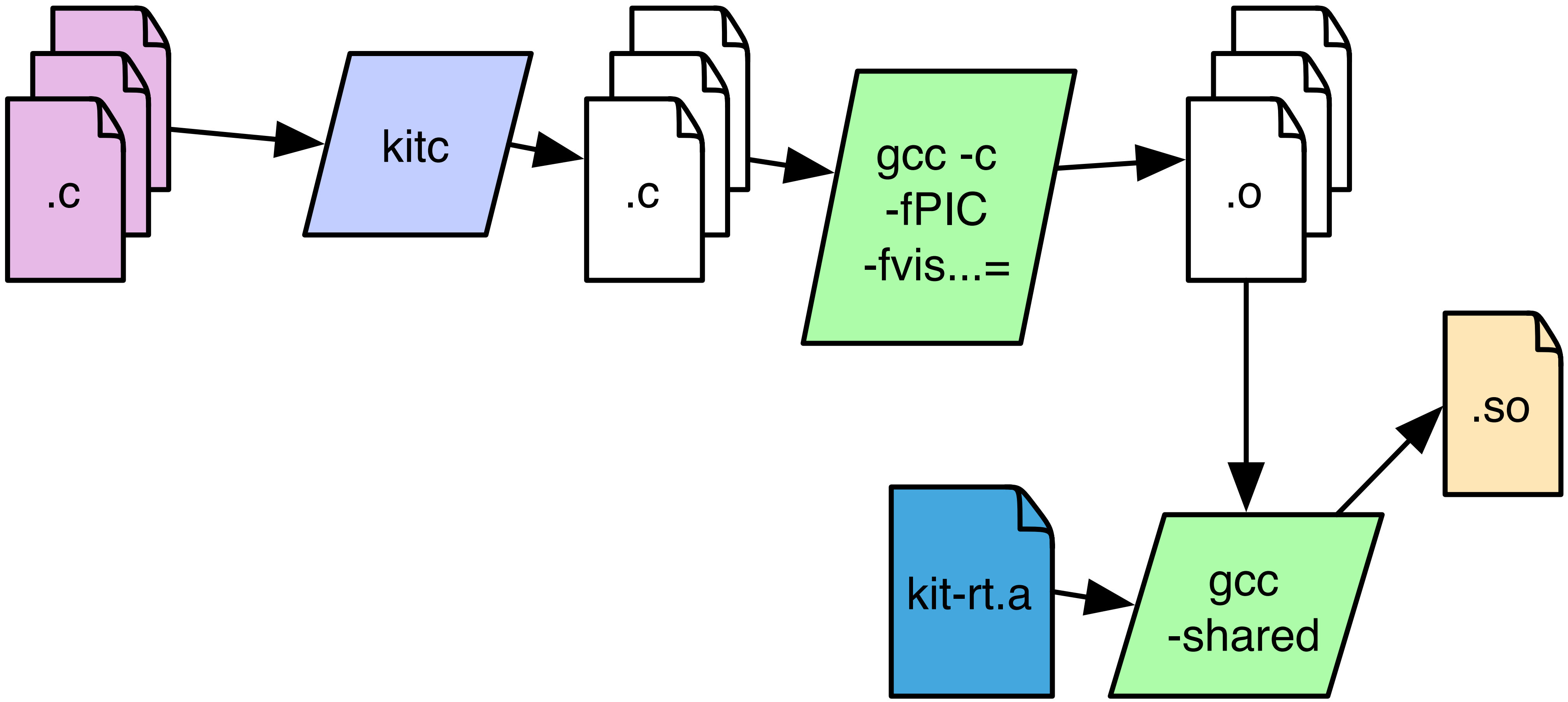
1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state
6. Free up old resources

Kitsune: whole-program updates



1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state
6. Free up old resources
7. Continue with new version

Kitsune build process



<u>Summary:</u>

- For each source file
  - replace gcc -c with composition of kitc and gcc
- Add -shared flag to linker and include kit-rt.a
- Allows us to update the entire program at once

Programmer obligations

- To implement DSU as a program feature, Kitsune requires the programmer to:
  - Choose update points: where updates may take place
  - Code for data migration: Identify the state to be transformed, and where it should be received in the new code
  - Code for control migration: Ensure execution reaches the right event loop when the new version restarts

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
       // ... process client requests
   }
}
int main() {
   mapping = malloc(...);
   l_fd = setup_conn();
   while (1) {
      client_loop();
   }
}
```

before modification

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {

     // ... process client requests
}  }
int main() {



    mapping = malloc(...);
    l_fd = setup_conn();
```

```
   while (1) {

     client_loop();
   }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {

     // ... process client requests
}  }
int main() {



    mapping = malloc(...);
    l_fd = setup_conn();
```

1. Choose update points

One per long running loop

```
  while (1) {

    client_loop();
  }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {

     // ... process client requests
}  }
int main() {



    mapping = malloc(...);
    l_fd = setup_conn();
```

1. Choose update points

One per long running loop

```
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {
     kitsune_update("client");
     // ... process client requests
}  }
int main() {



    mapping = malloc(...);
    l_fd = setup_conn();
```

```
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

1. Choose update points

One per long running loop

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;
int l_fd;
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {
     kitsune_update("client");
     // ... process client requests
}  }
int main() {


    mapping = malloc(...);
    l_fd = setup_conn();
```

2. Add data migration code

Globals migrated by default

Initiate at start of main()

```
   while (1) {
     kitsune_update("main");
     client_loop();
   }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;  // automigrated
int l_fd;       // automigrated
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {
     kitsune_update("client");
     // ... process client requests
}  }
int main() {
  kitsune_do_automigrate();

    mapping = malloc(...);
    l_fd = setup_conn();
```

2. Add data migration code

Globals migrated by default

Initiate at start of main()

```
   while (1) {
     kitsune_update("main");
     client_loop();
   }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;  // automigrated
int l_fd;       // automigrated
void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    kitsune_update("client");
    // ... process client requests
}  }
int main() {
  kitsune_do_automigrate();

   mapping = malloc(...);
   l_fd = setup_conn();
```

3. Add control migration code

Avoid reinitialization

Redirect control to update point

```
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;  // automigrated
int l_fd;       // automigrated
void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    kitsune_update("client");
    // ... process client requests
} }
int main() {
 kitsune_do_automigrate();
 if (!kitsune_is_updating()) {
   mapping = malloc(...);
   l_fd = setup_conn();
 }
```

3. Add control migration code

Avoid reinitialization

Redirect control to update point

```
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

after modification for Kitsune

Example single-threaded server

```
typedef int data;
data *mapping;  // automigrated
int l_fd;       // automigrated
void client_loop() {
   int cl_fd = get_conn(l_fd);
   while (1) {
     kitsune_update("client");
     // ... process client requests
}  }
int main() {
  kitsune_do_automigrate();
  if (!kitsune_is_updating()) {
    mapping = malloc(...);
    l_fd = setup_conn();
  }
```

3. Add control migration code

Avoid reinitialization

Redirect control to update point

```
  if (kitsune_is_updating_from
        ("client")) {
    client_loop();
  }
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

after modification for Kitsune

Example single-threaded server

We also support migration of locals
Generalizes to multi-threaded programs

```
typedef int data;
data *mapping;  // automigrated
int l_fd;       // automigrated
void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    kitsune_update("client");
    // ... process client requests
} }
int main() {
 kitsune_do_automigrate();
 if (!kitsune_is_updating()) {
   mapping = malloc(...);
   l_fd = setup_conn();
 }
  if (kitsune_is_updating_from
      ("client")) {
    client_loop();
  }
  while (1) {
    kitsune_update("main");
    client_loop();
  }
}
```

after modification for Kitsune

Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

```
typedef char *data;
data *mapping;
```

new

Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

```
typedef char *data;
data *mapping;
```

new

Xform

For each value **x** *of type* **data** *in the running program*
and its corresponding location **p** *in the new program*
do
  ***p = malloc(N);**
  snprintf(*p,N,"%d",x);
end

Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

```
typedef char *data;
data *mapping;
```

new

Xform

```
new::mapsz = old::mapsz;
new::mapping = malloc(new::mapsz*sizeof(char*));
for (int i=0;i<new::mapsz;i++) {
  old::data x = old::mapping[i];
  new::data *p = &new::mapping[i];
  *p = malloc(N);
  snprintf(*p,N,"%d",x);
}
```

Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

```
typedef char *data;
data *mapping;
```

new

Xform

Xfgen tool

- Require programmer to write *relevant* xform code using high-level specs
- Automate generation of transformation code
  - requires some additional type annotations

Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

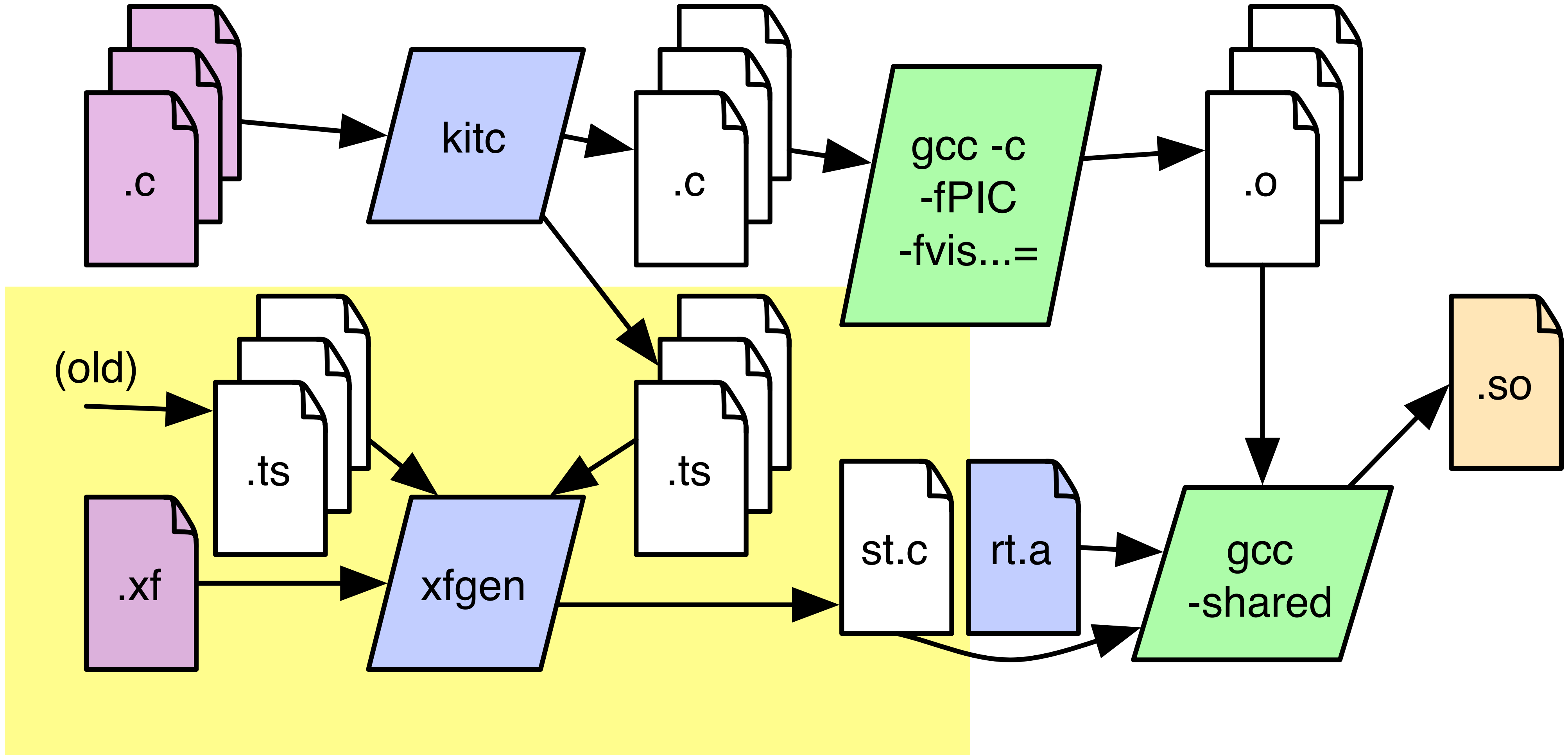
```
typedef char *data;
data *mapping;
```

new

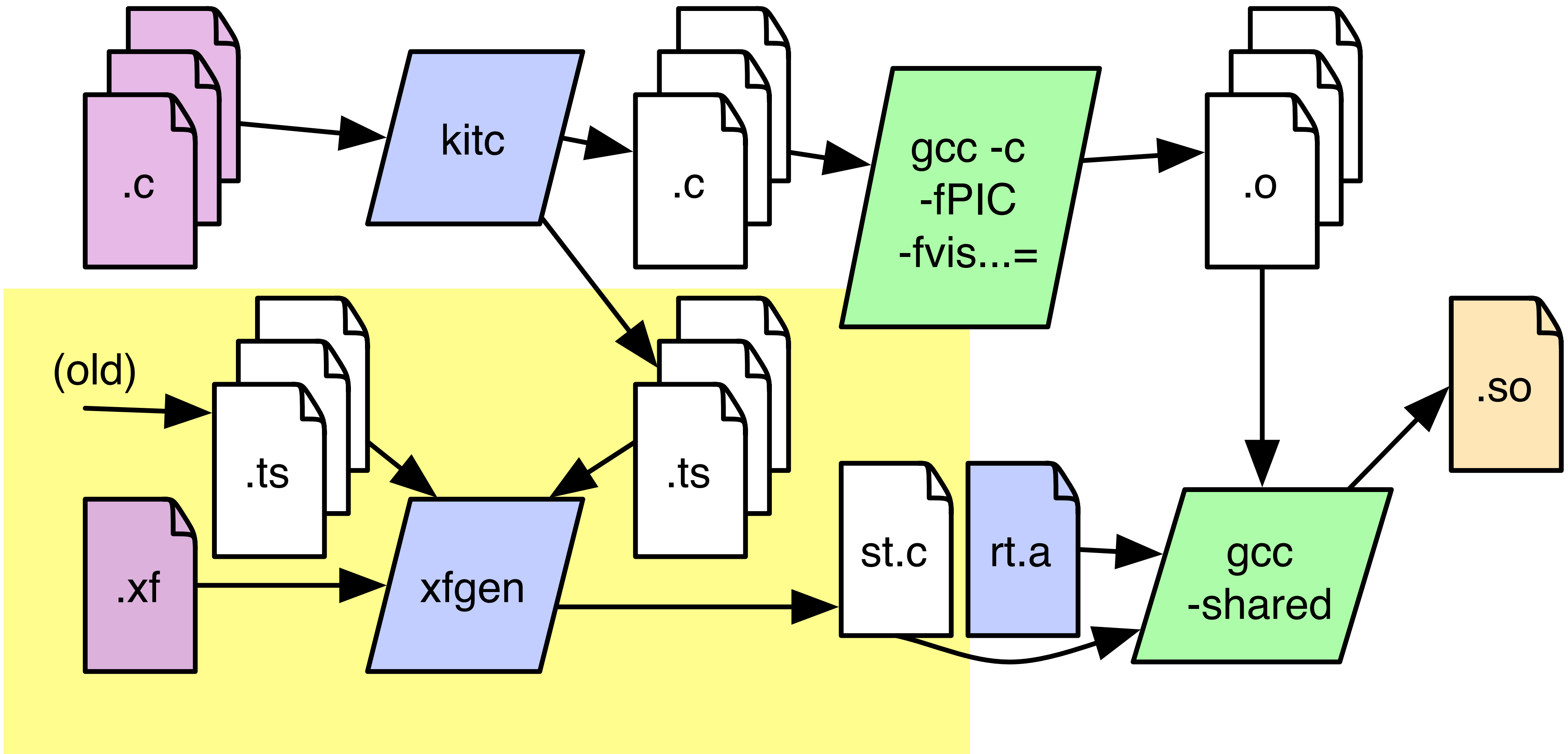
Xform

```
typedef data → typedef data: {
    $out = malloc(N);
    snprintf($out, N, "%d", $in);
}
```

Using Kitsune and xfgen



Using Kitsune and xfgen



- Transformation specs in per-update .xf file
- Linked in with new version and invoked by kitsune_do_automigrate() and MIGRATE_LOCAL()

Kitsune benchmarks: changes required

Kitsune benchmarks: changes required

Program	# Vers		LoC
vsftpd	14	(1.1.0–2.0.6)	12,202
redis	5	(2.0.0–2.0.4)	13,387
Tor	13	(0.2.1.18–0.2.1.30)	76,090
*memcached**	3	(1.2.2–1.2.4)	4,181
*icecast**	5	(2.2.0–2.3.1)	15,759
*snort**	4	(2.9.2–2.9.2.3)	214,703

*Multi-threaded

Kitsune benchmarks: changes required

Program	# Vers		LoC
vsftpd	14	(1.1.0–2.0.6)	12,202
redis	5	(2.0.0–2.0.4)	13,387
Tor	13	(0.2.1.18–0.2.1.30)	76,090
*memcached**	3	(1.2.2–1.2.4)	4,181
*icecast**	5	(2.2.0–2.3.1)	15,759
*snort**	4	(2.9.2–2.9.2.3)	214,703

*Multi-threaded

Program	Upd	Ctrl	Data	E_*	Oth	Σ	v→v	t→t	Σ	xf LoC
vsftpd	6	26	17+8	6+14	28+8	83+30	9	21	30	101
redis	1	2	3	43	8	57	0	4	4	37
Tor	1	39	37+6	19	57	153+6	16	15	31	189
*memcached**	4	9	13	20	66	112	12	10	22	27
*icecast**	11+1	22+3	14+9	32+3	39	118+16	25	50	75	200
*snort**	2	90+18	110+2	158	66	426+20	111	64	175	197

Performance overhead

Program	Orig (siqr)	Kitsune	Ginseng	UpStare
64-bit, 4×2.4Ghz E7450 (6 core), 24GB mem, RHEL 5.7				
vsftpd 2.0.6*	6.55s (0.04s)	+0.75%	–	–
memchd 1.2.4	59.30s (3.25s)	+0.51%	–	–
redis 2.0.4	46.83s (0.40s)	-0.31%	–	–
icecast 2.3.1	10.11s (2.27s)	-2.18%	–	–
32-bit, 1×3.6Ghz Pentium D (2 core), 2GB mem, Ubuntu 10.10				
vsftpd 2.0.3*	5.96s (0.01s)	+2.35%	+11.3%	+41.6%
vsftpd 2.0.3†	14.03s (0.02s)	+0.29%	+1.47%	+6.64%
memchd 1.2.4	101.40s (0.35s)	-0.49%	+18.4%	–
redis 2.0.4	43.88s (0.16s)	-1.21%	–	–
icecast 2.3.1	35.71s (0.68s)	+1.18%	-0.28%	–

*CD+LS benchmark, †file download benchmark

- 21 runs each, median, siqr reported
- Overall: -2.18% to 2.35% overhead (in the noise)
- (No performance measurements for snort yet)

Update times

Program	Med.	(siqr)	Min	Max
64-bit, 4×2.4Ghz E7450 (6 core), 24GB mem, RHEL 5.7				
vsftpd →2.0.6	2.99ms	(0.04ms)	2.62	3.09
memcached →1.2.4	2.50ms	(0.05ms)	2.27	2.68
redis →2.0.4	39.70ms	(0.98ms)	36.14	82.66
icecast →2.3.1	990.89ms	(0.95ms)	451.73	992.71
icecast-nsp →2.3.1	187.89ms	(1.77ms)	87.14	191.32
tor →0.2.1.30	11.81ms	(0.12ms)	11.65	13.83
32-bit, 1×3.6Ghz Pentium D (2 core), 2GB mem, Ubuntu 10.10				
vsftpd →2.0.3	2.62ms	(0.03ms)	2.52	2.71
memcached →1.2.4	2.44ms	(0.08ms)	2.27	3.12
redis →2.0.4	38.83ms	(0.64ms)	37.69	41.80
icecast →2.3.1	885.39ms	(7.47ms)	859.00	908.87
tor →0.2.1.30	10.43ms	(0.46ms)	10.08	12.98

- < 40ms in all cases but icecast
  - Icecast includes 1s sleeps; icecast-nsp removes these

Key idea #1: Update points

Key idea #1: Update points

- Competing approach: update anywhere
  - (when code to be changed not running)
  - Used by Ksplice, K42 (OS), OPUS

Key idea #1: Update points

- Competing approach: update anywhere
  - (when code to be changed not running)
  - Used by Ksplice, K42 (OS), OPUS
- Benefits of update points
  - Simplifies reasoning for programmers
    - Particularly for multithreaded programs
  - May accelerate update times
    - As opposed to waiting for updated code to become inactive
  - Simplifies updating mechanism

Key idea #2: Whole program updates

Key idea #2: Whole program updates

- Competing approach
  - Program keeps running the current code, and subsequent function calls to new versions
  - Used by Ginseng, POLUS, OPUS, Ksplice, K42

Key idea #2: Whole program updates

- Competing approach
  - Program keeps running the current code, and subsequent function calls to new versions
  - Used by Ginseng, POLUS, OPUS, Ksplice, K42
- Benefits of whole-program updates:
  - Can update active code (e.g., long-running loops) in an arbitrary manner
    - very important in practice
  - Explicit control migration simplifies reasoning, maintenance
  - More efficient implementation
    - No need to insert levels of indirection, use trampolines, etc.
    - No need to compile datastructures differently

Ongoing work

Ongoing work

- Means to specify and verify the correctness of dynamic software updates [VSTTE'12]
  - Reuse specifications for each version individually
  - Explicate acceptable backward-incompatible behaviors

Ongoing work

- Means to specify and verify the correctness of dynamic software updates [VSTTE'12]
  - Reuse specifications for each version individually
  - Explicate acceptable backward-incompatible behaviors
- Means to automatically generate state transformations from dynamic analysis [OOPSLA'12]
  - E.g., automatically correct leaks in running heap

Ongoing work

- Means to specify and verify the correctness of dynamic software updates [VSTTE'12]
  - Reuse specifications for each version individually
  - Explicate acceptable backward-incompatible behaviors
- Means to automatically generate state transformations from dynamic analysis [OOPSLA'12]
  - E.g., automatically correct leaks in running heap
- Adapt Kitsune methodology to Java
  - Contrast to our earlier VM-based approach [PLDI'09]

Ongoing work

- Means to specify and verify the correctness of dynamic software updates [VSTTE'12]
  - Reuse specifications for each version individually
  - Explicate acceptable backward-incompatible behaviors
- Means to automatically generate state transformations from dynamic analysis [OOPSLA'12]
  - E.g., automatically correct leaks in running heap
- Adapt Kitsune methodology to Java
  - Contrast to our earlier VM-based approach [PLDI'09]
- Implement lazy state transformation for Kitsune

DSU project team

- Former students / post-docs
  - Manuel Oriol, post-doc 2005-06, @University of York (UK) and ABB
  - Gareth Stoyle, Ph.D. (Cambridge) 2007, @UBS (UK)
  - Iulian Neamtiu, Ph.D. 2008, @UC Riverside
  - Suriya Subramanian, Ph.D. (UT Austin) 2011, @Intel
  - Stephen Magill, post-doc 2010-11, @IDA/CCS (Gov. lab)
  - Chris Hayden, Ph.D. 2012, @Washington Post Labs
- Current students
  - Karla Saur (3rd year), Ted Smith (undergrad), Luis Pina (3rd year, visiting)
- Profs/researchers
  - Kathryn McKinley, Prof @UT, MSR; Jeff Foster, Prof @Maryland;
  - Nate Foster, Prof @Cornell; Peter Sewell, Prof @Cambridge; Gavin Bierman, @MSR Cambridge

Roadmap

- Dynamic software updating (DSU)
  - Kitsune: Flexible and Efficient DSU for C programs
- Program analysis for security and reliability
  - Knowledge-based security: quantitatively tracking information
- Quick tour of some other work

Program analysis to improve quality

- Software is ubiquitous, and critically important
  - Yet it is often unreliable and insecure
- So: build tools to analyze software automatically
  - **Static analysis** applied before running the program
    - Examples: Type checkers/inferencers, tools like FindBugs
    - Pros: Complete coverage (considers all runs), no run-time overhead
    - Cons: problems are undecidable, so often false alarms
  - **Dynamic analysis** observes actual executions
    - Pros: Very precise, no false alarms
    - Cons: Less coverage, instrumentation adds run-time overhead, discovered problems hard to remediate in deployment

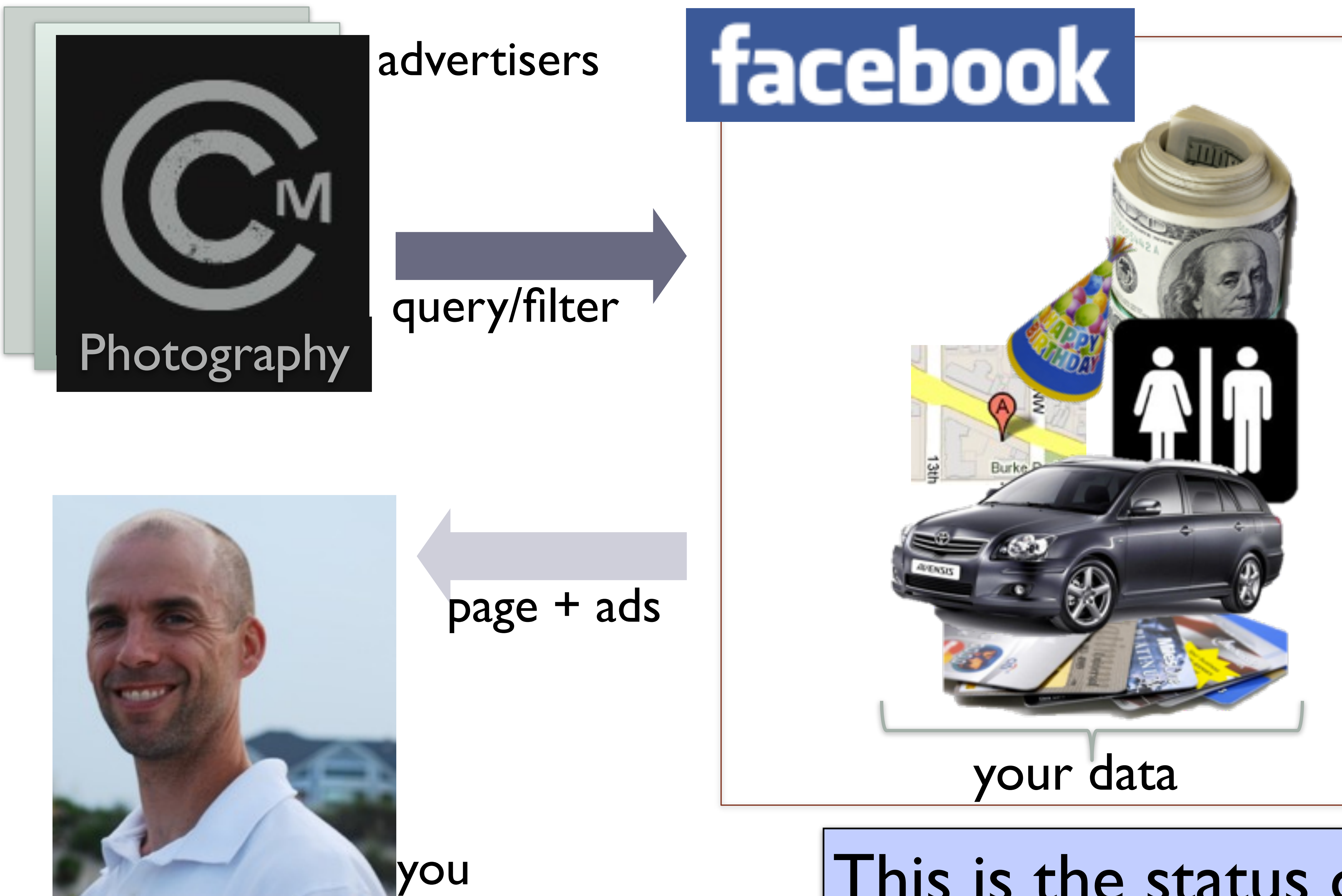
Hybrid analysis: best of both worlds

- Dynamic analysis, ***optimized*** by static analysis
  - Eliminate redundant checks; no false alarms
  - *Ex: concurrency error checking* [POPL'10], *atomicity enforcement* [TX'06]
- Dynamic analysis, ***proved correct*** statically
  - Prove that necessary checks take place for all possible executions
  - *Ex: Fable/SELinks for security checking* [Oakland'08, SIGMOD'09]
- Static analysis, ***made more precise*** by dynamic analysis
  - Added contextual information reduces false alarms
  - *Ex: Synthesis of DSU state transformers* [OOPSLA'12], *Knowledge-based security* [CSF'11, PLAS'12], *Rubydust* [POPL'11, STOP'11]

Hybrid analysis: best of both worlds

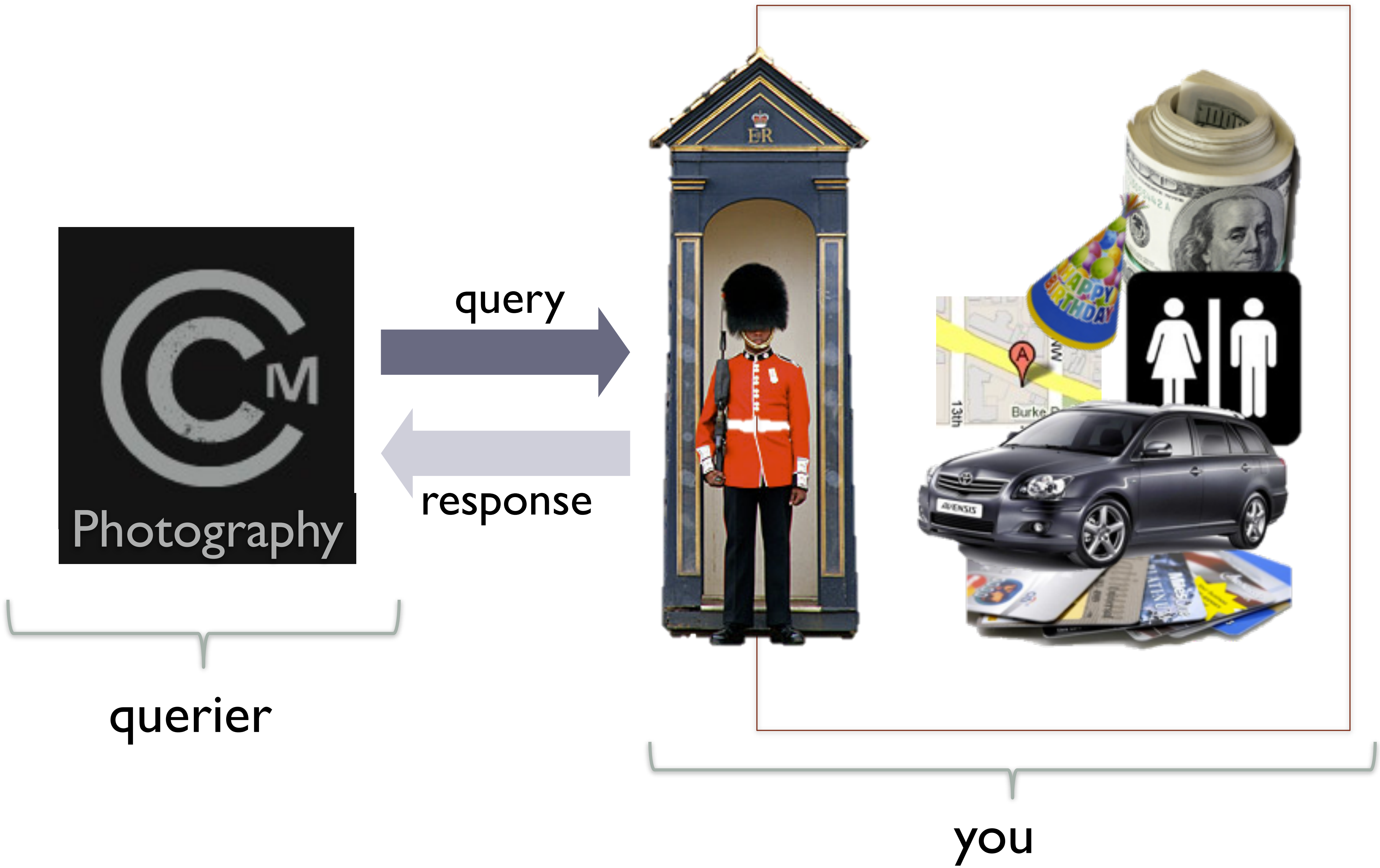
- Dynamic analysis, ***optimized*** by static analysis
  - Eliminate redundant checks; no false alarms
  - *Ex: concurrency error checking* [POPL'10], *atomicity enforcement* [TX'06]
- Dynamic analysis, ***proved correct*** statically
  - Prove that necessary checks take place for all possible executions
  - *Ex: Fable/SELinks for security checking* [Oakland'08, SIGMOD'09]
- Static analysis, ***made more precise*** by dynamic analysis
  - Added contextual information reduces false alarms
  - *Ex: Synthesis of DSU state transformers* [OOPSLA'12], *Knowledge-based security* [CSF'11, PLAS'12], *Rubydust* [POPL'11, STOP'11]

No privacy: They have your data



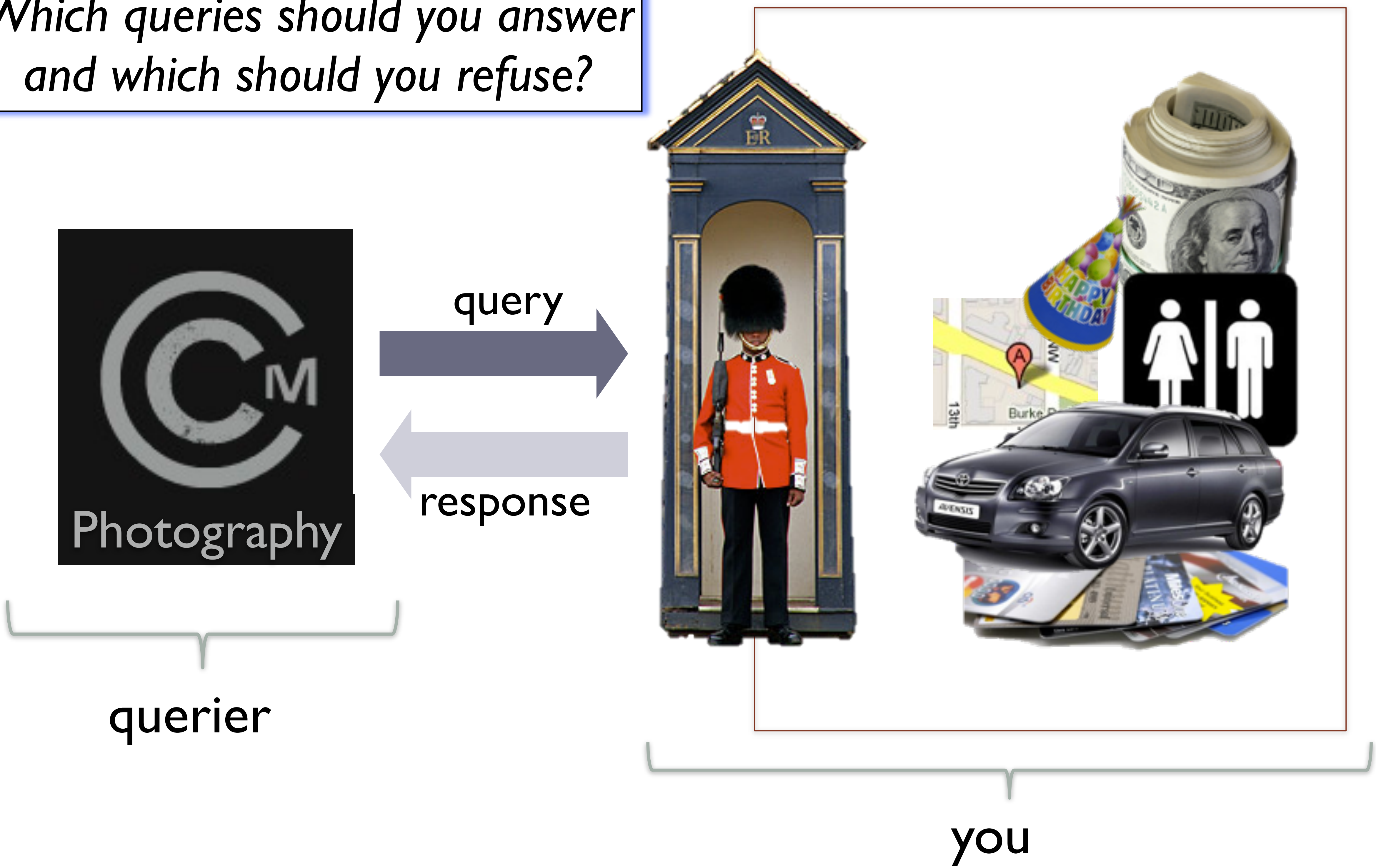
This is the status quo

Alternative: Maintain your own data



Alternative: Maintain your own data

The question then becomes:
Which queries should you answer and which should you refuse?



Query 1: Useful and non-revealing

out = 24 ≤ Age ≤ 30
& Female?
& Engaged? *

CM Photography

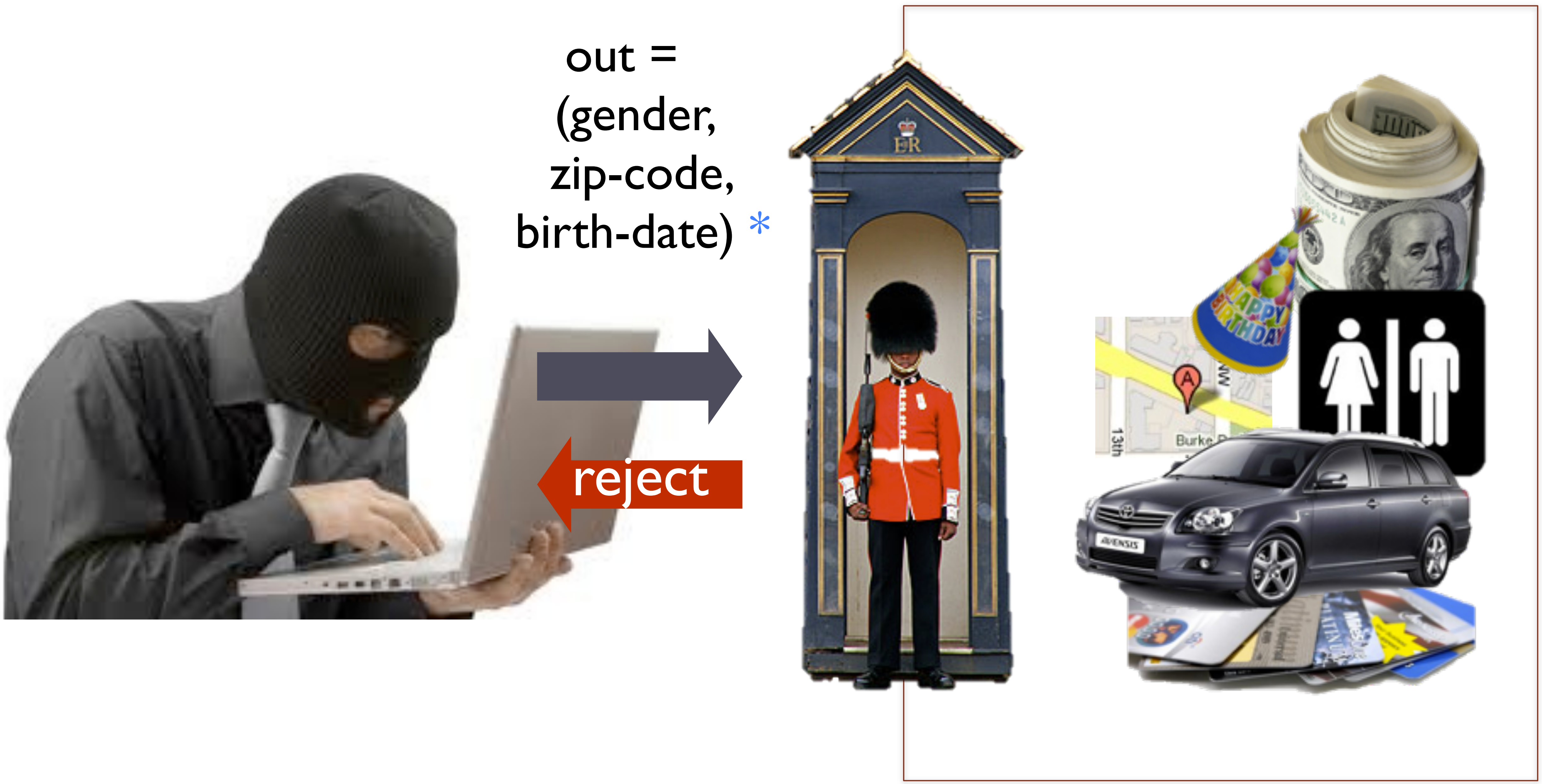
true

querier

you

* real query used by a Facebook advertiser

Query 2: Reveals too much!



* - gender, zip-code, birth-date can be used to uniquely identify 87% of Americans

When to accept, when to reject

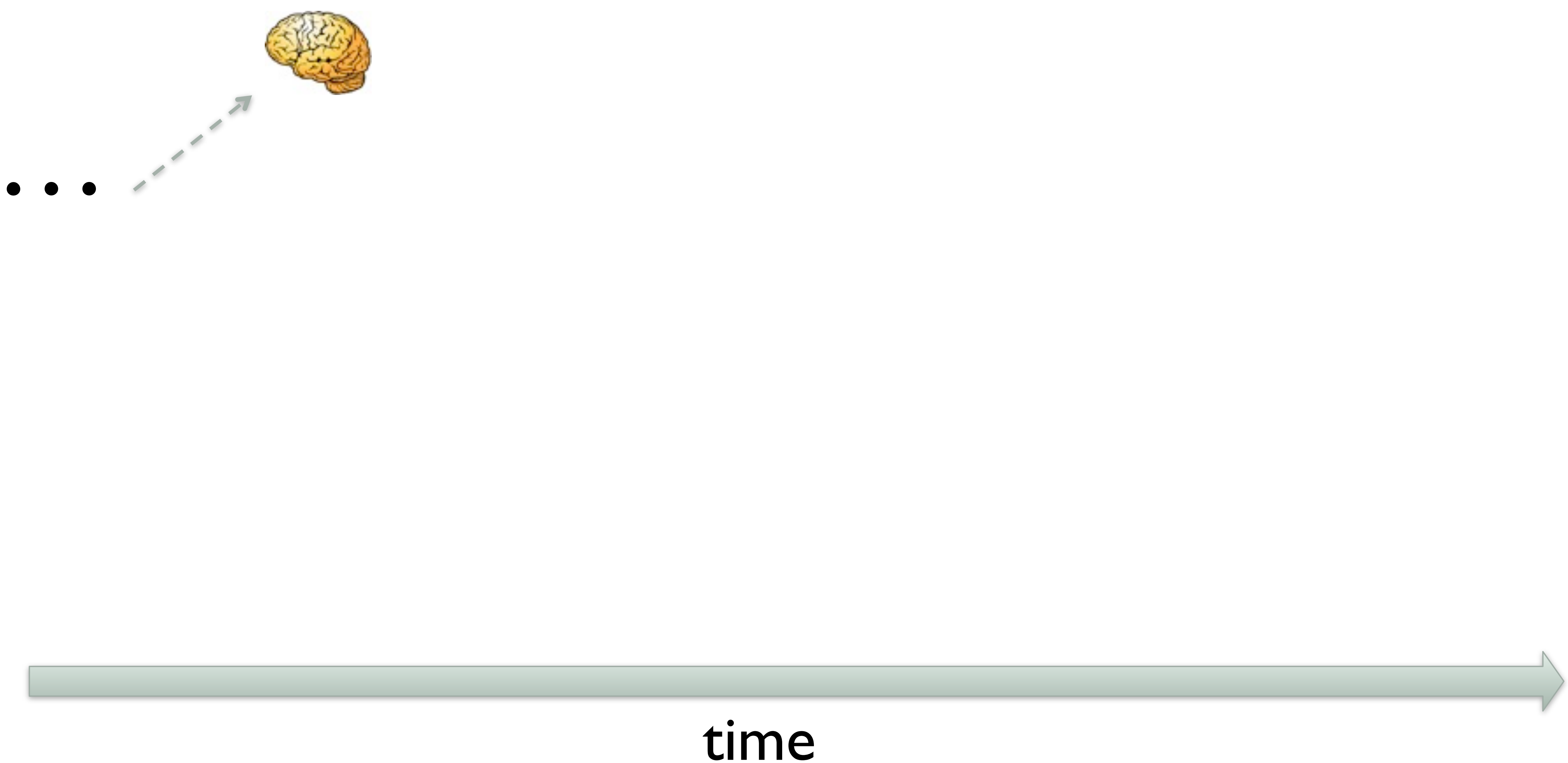
- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

• • •

time

When to accept, when to reject

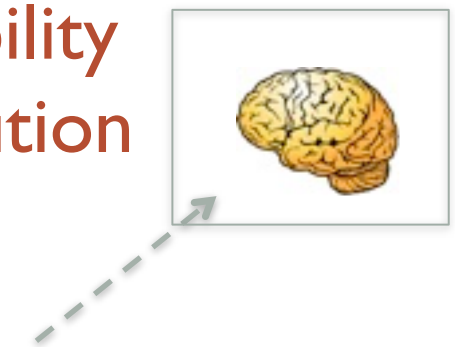
- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

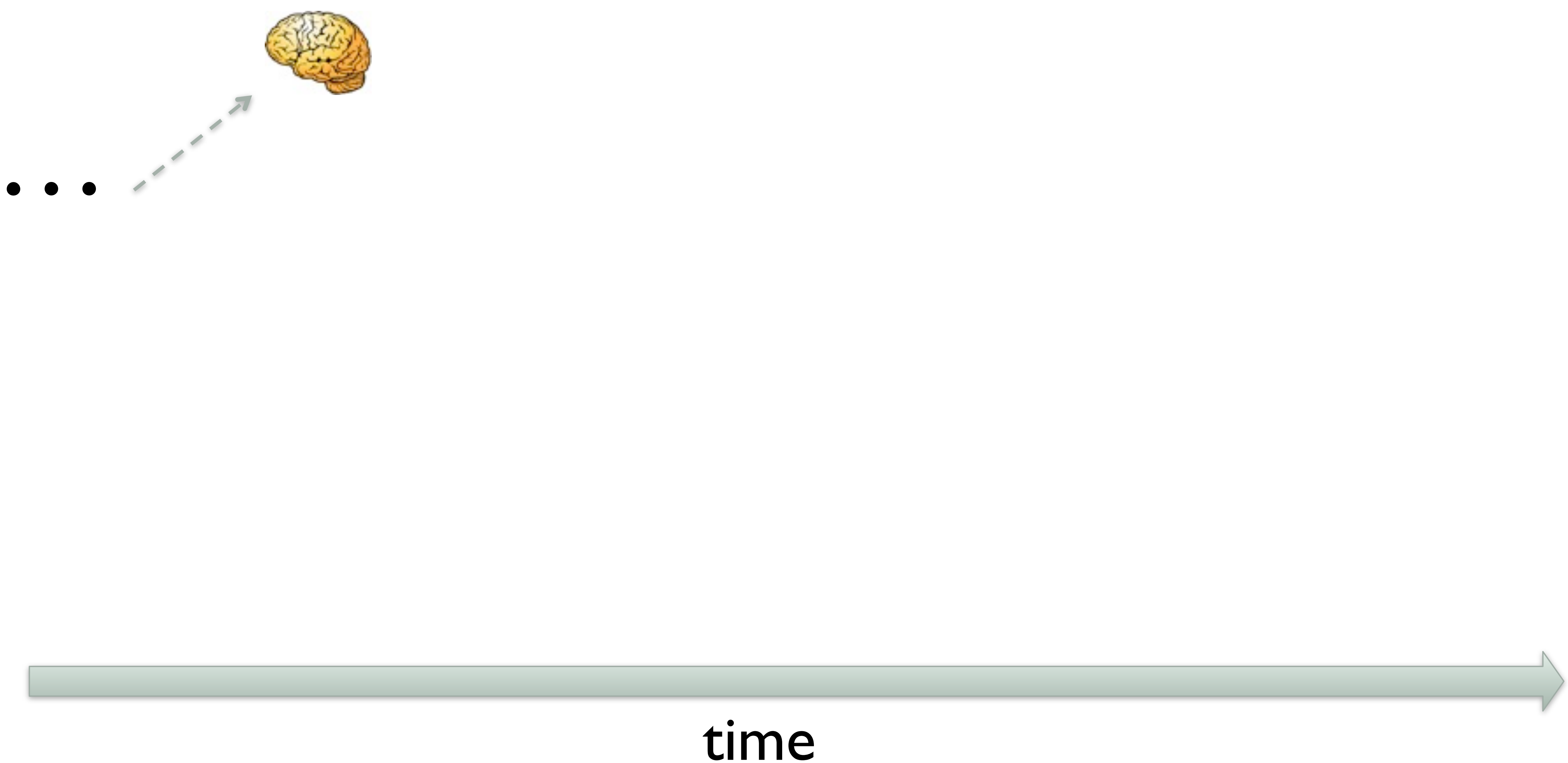
Belief ≜ probability distribution



. . .

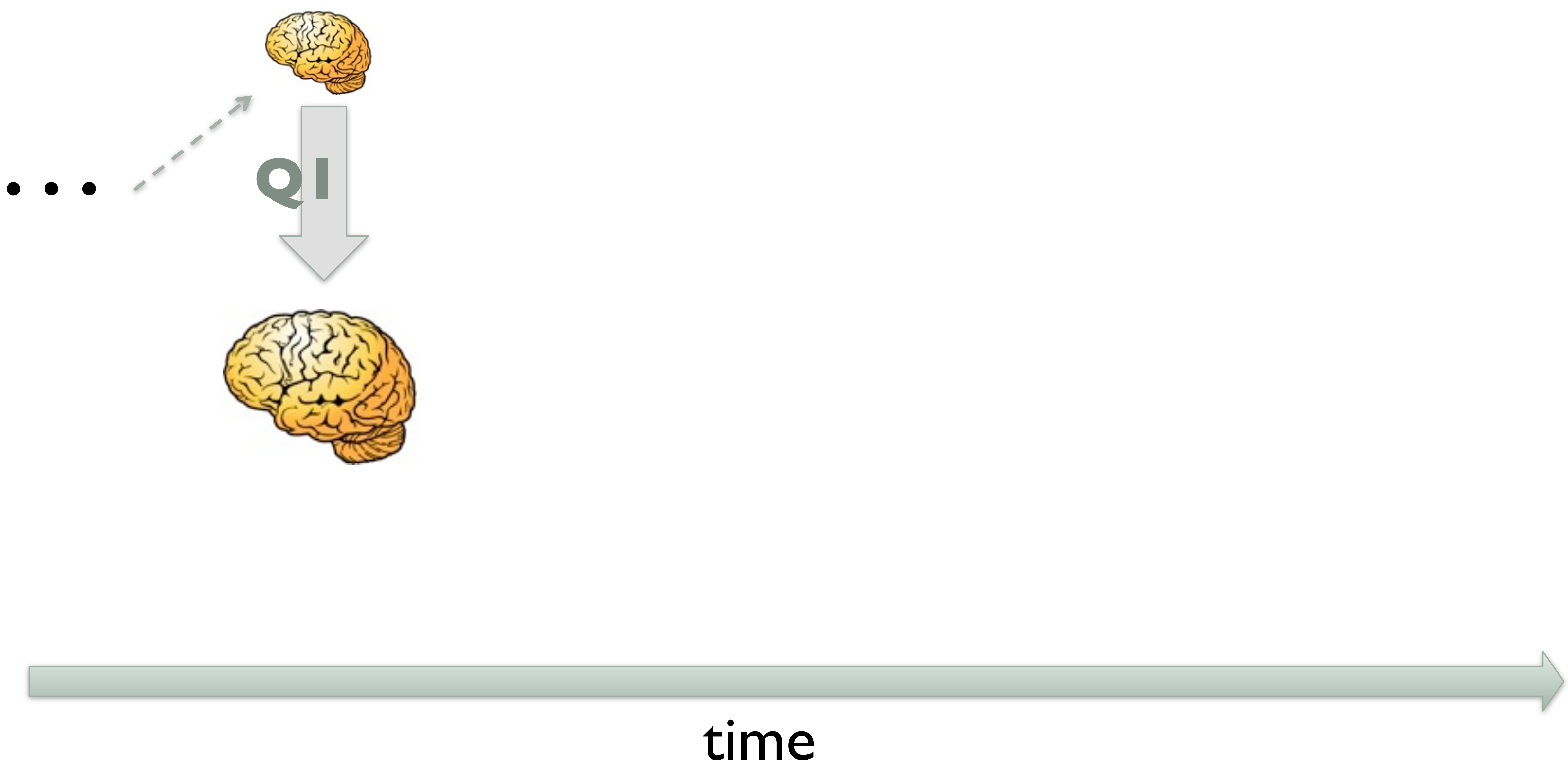
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



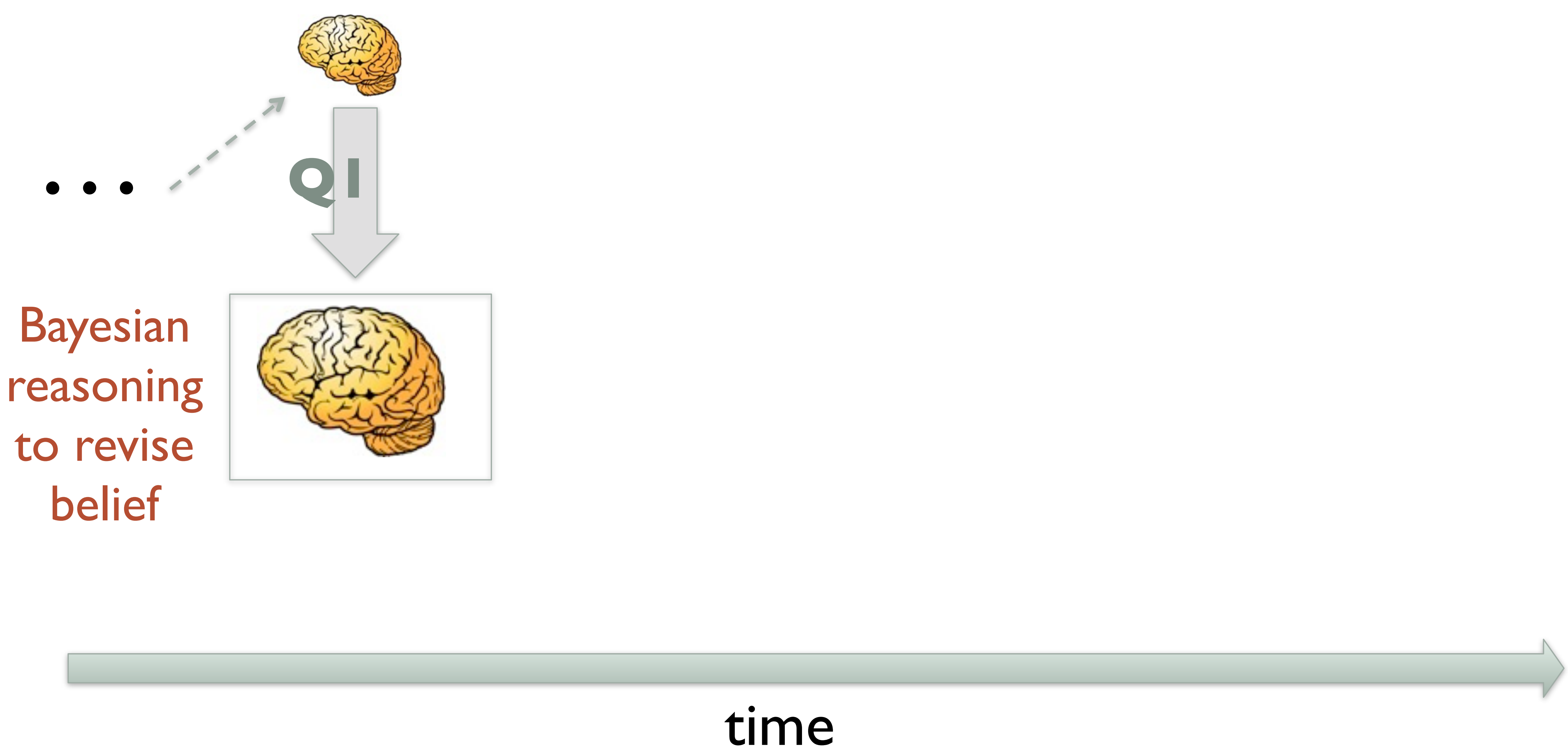
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

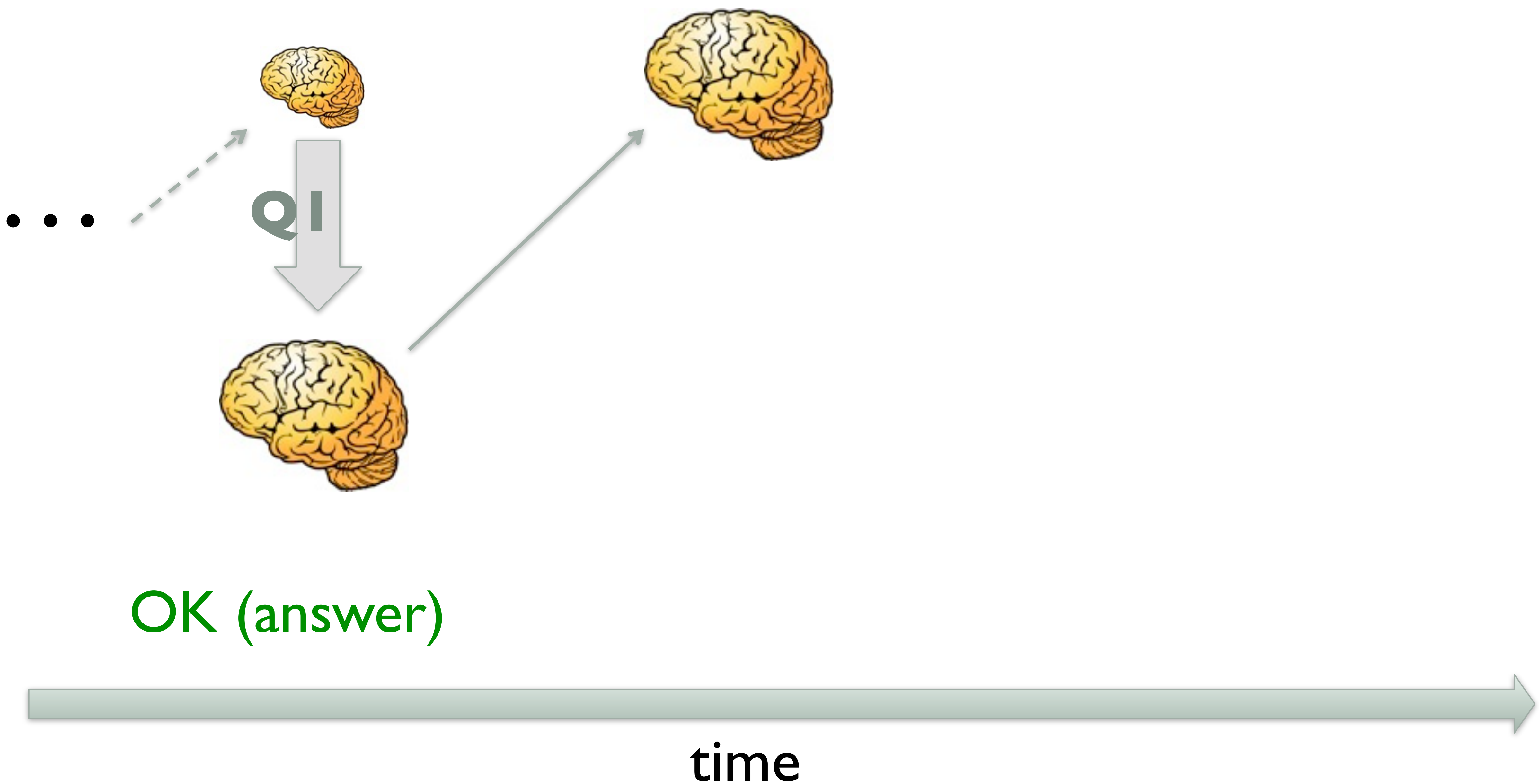


When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
    - Cannot let rejection defeat our protection.

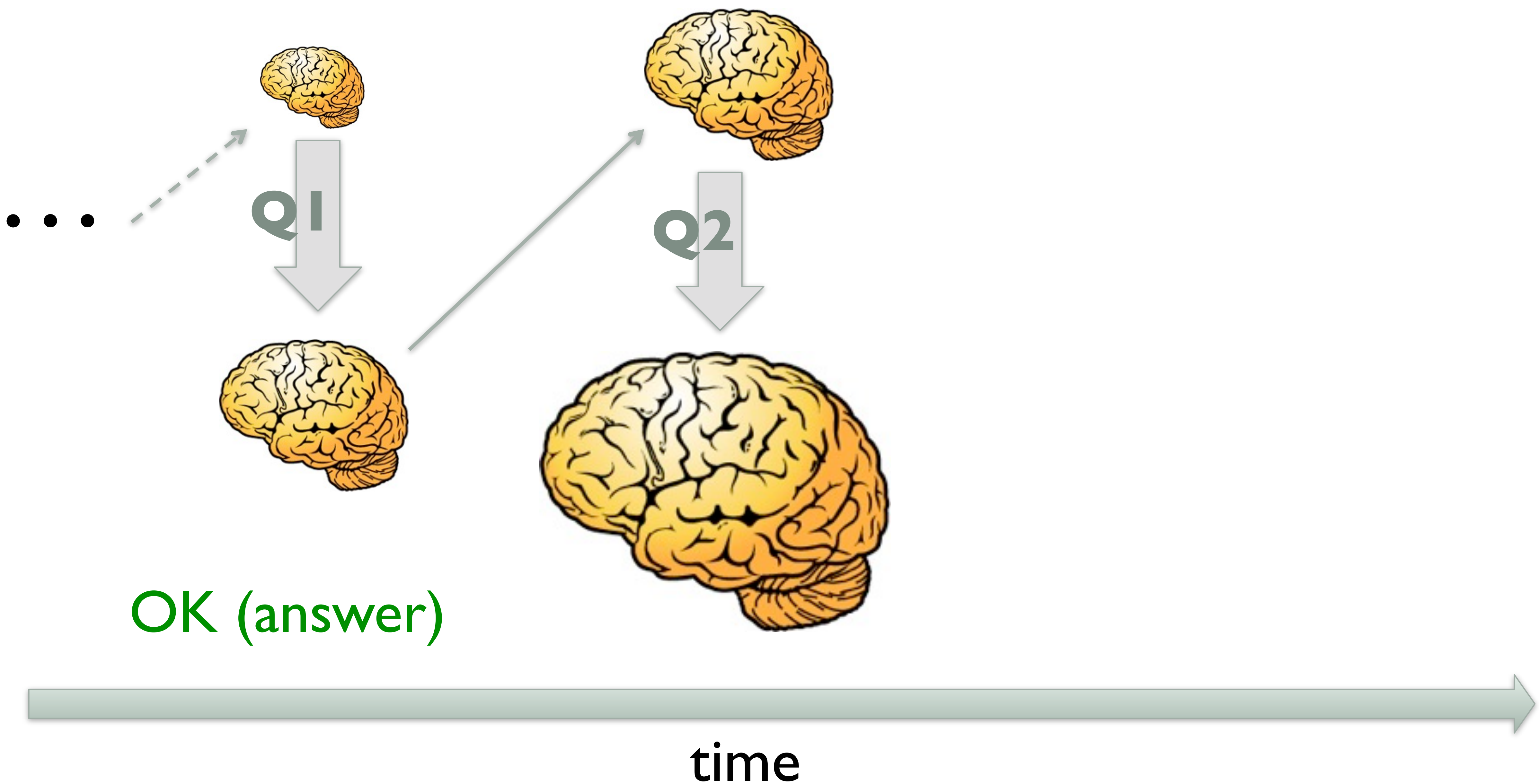
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



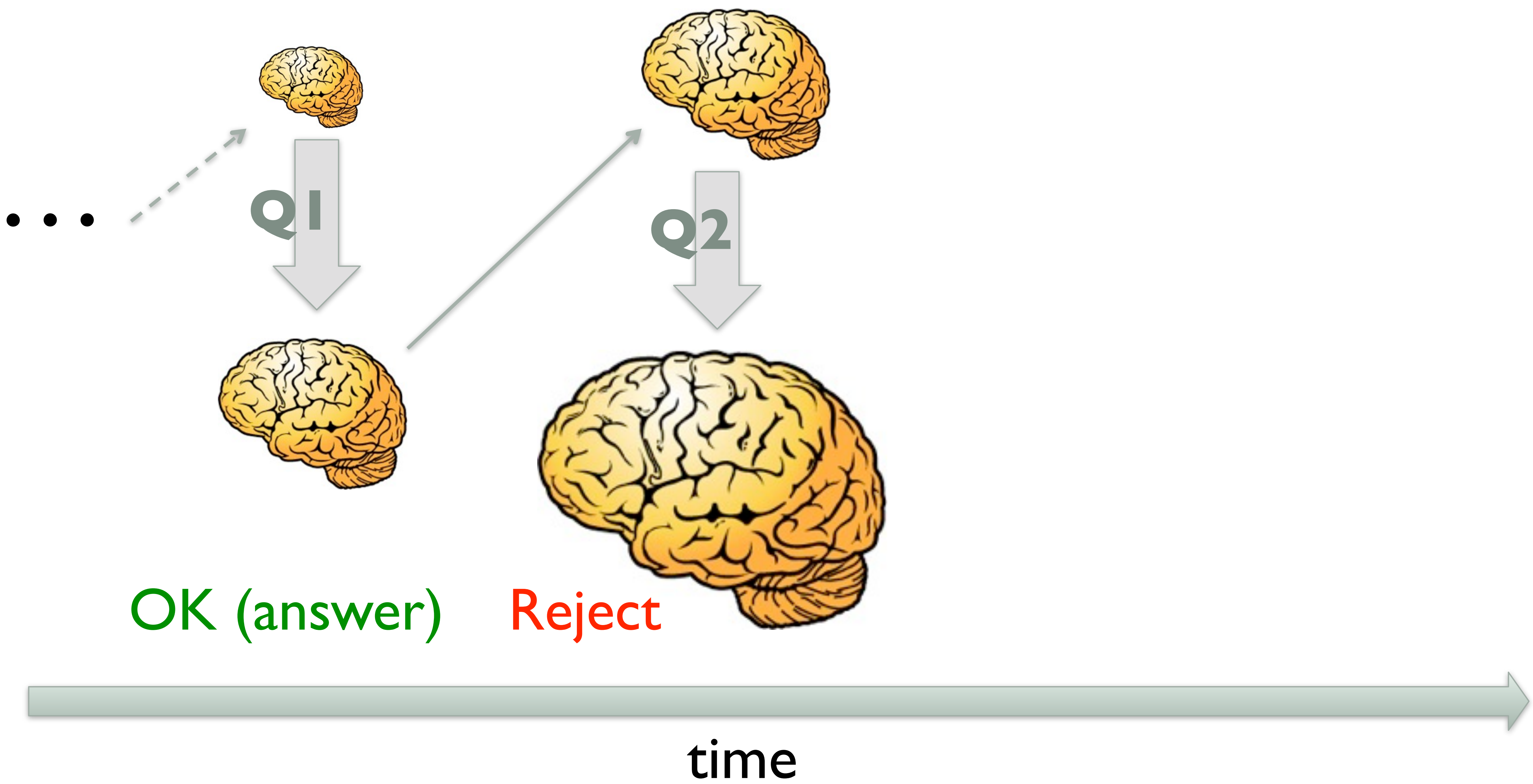
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



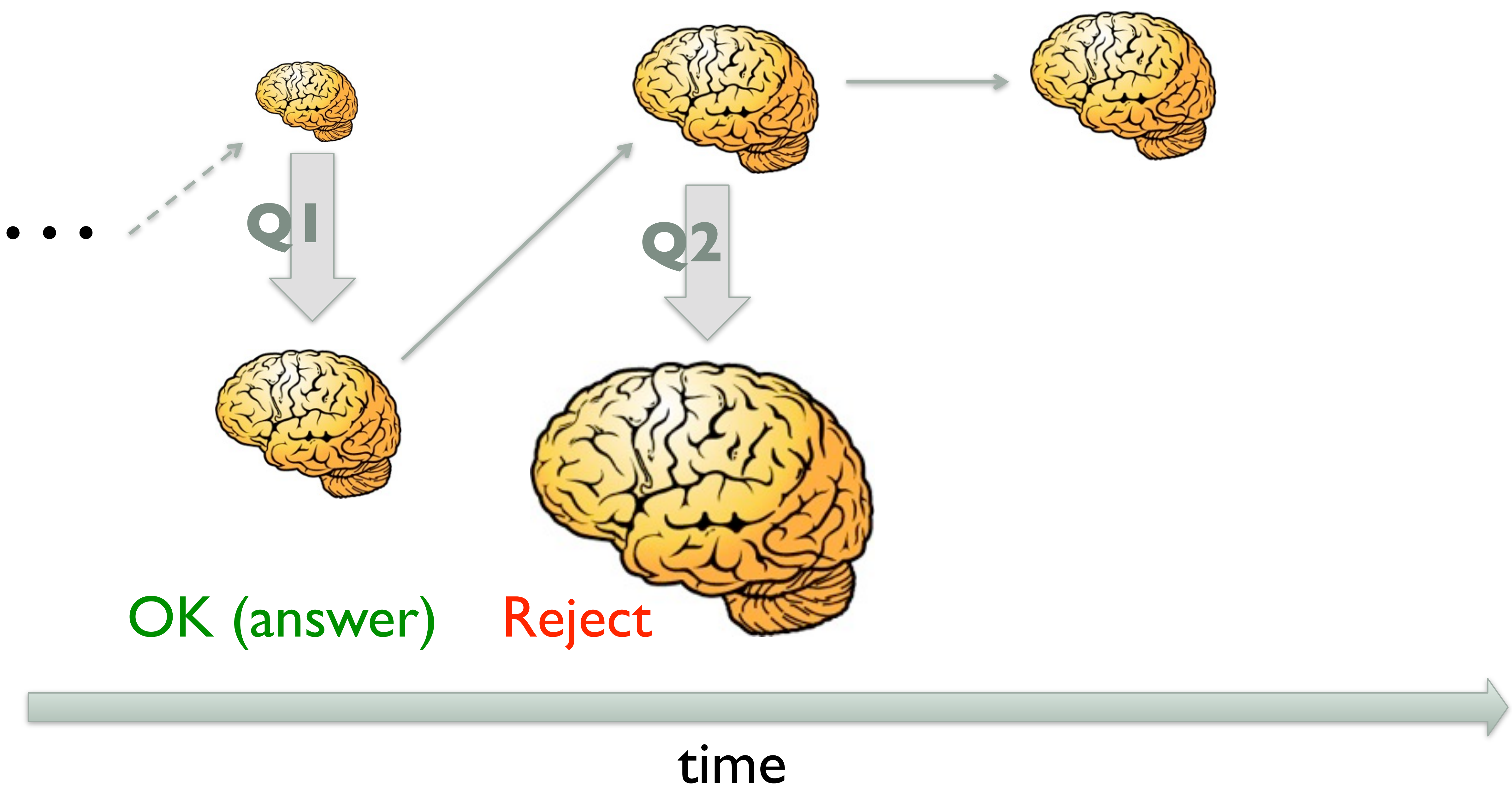
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



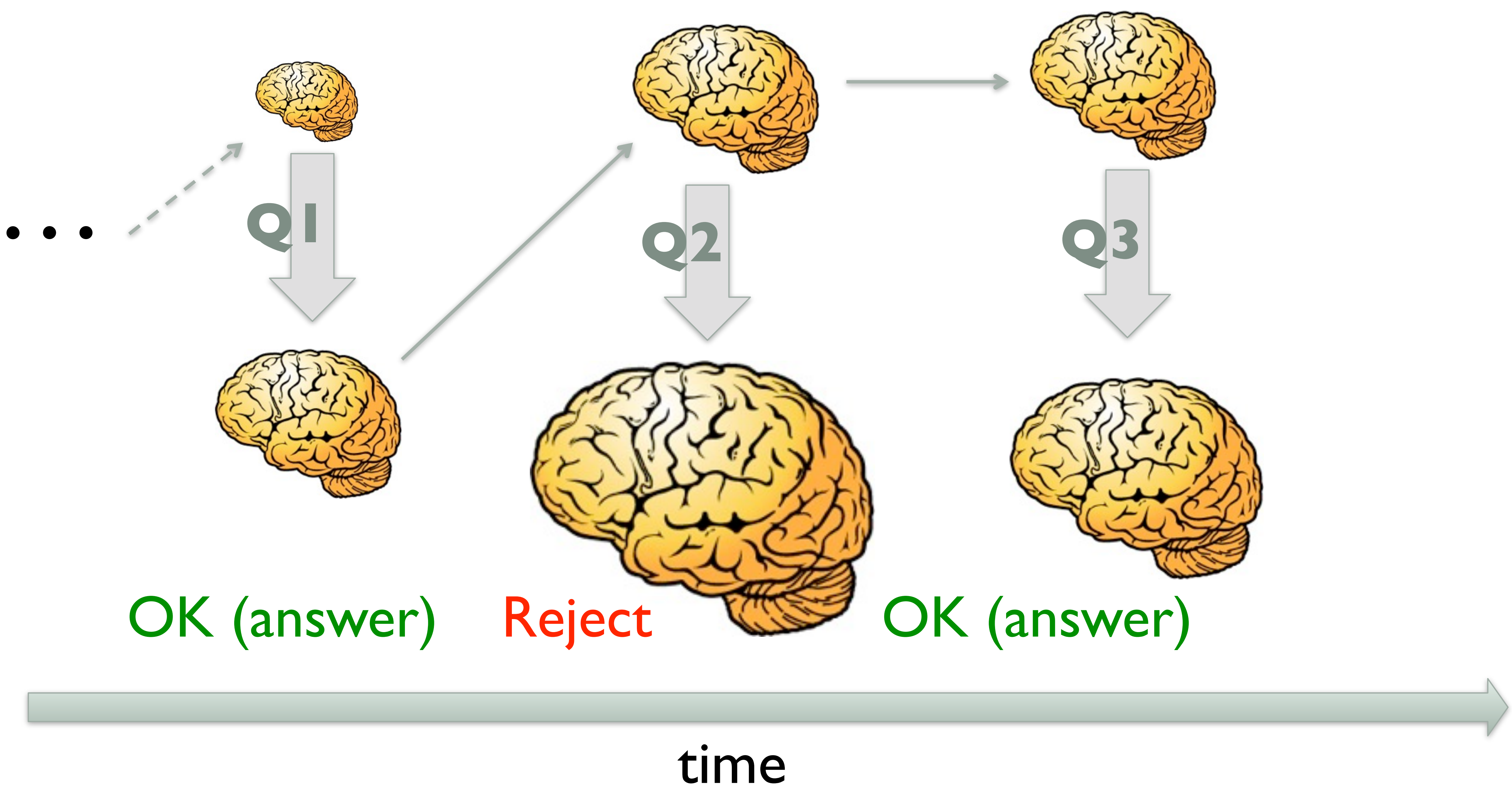
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



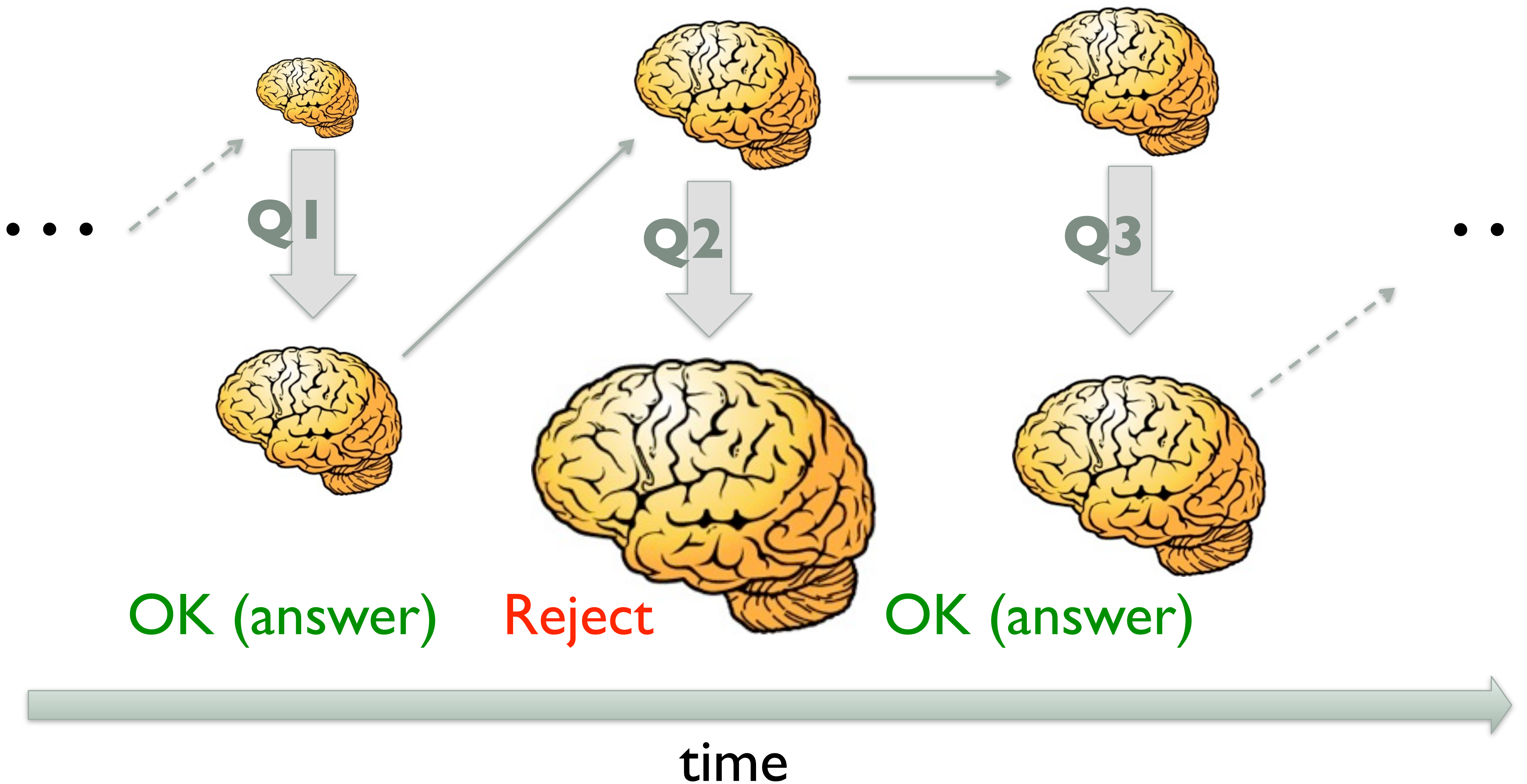
When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



When to accept, when to reject

- Maintain a representation of the querier's **belief** about secret's possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.



Meet Bob

Bob (born September 24, 1980)

bday = 267

byear = 1980

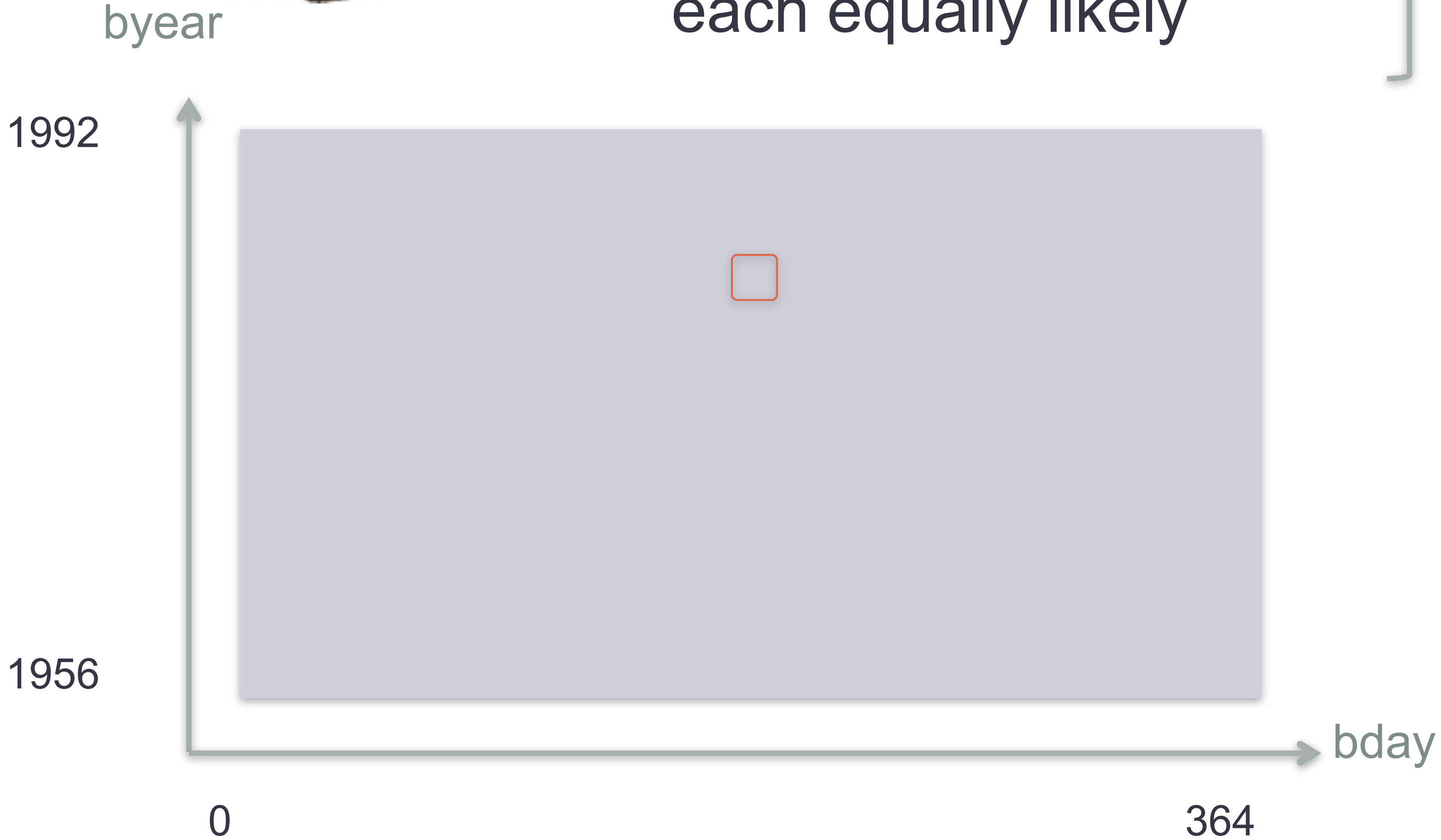
} Secret

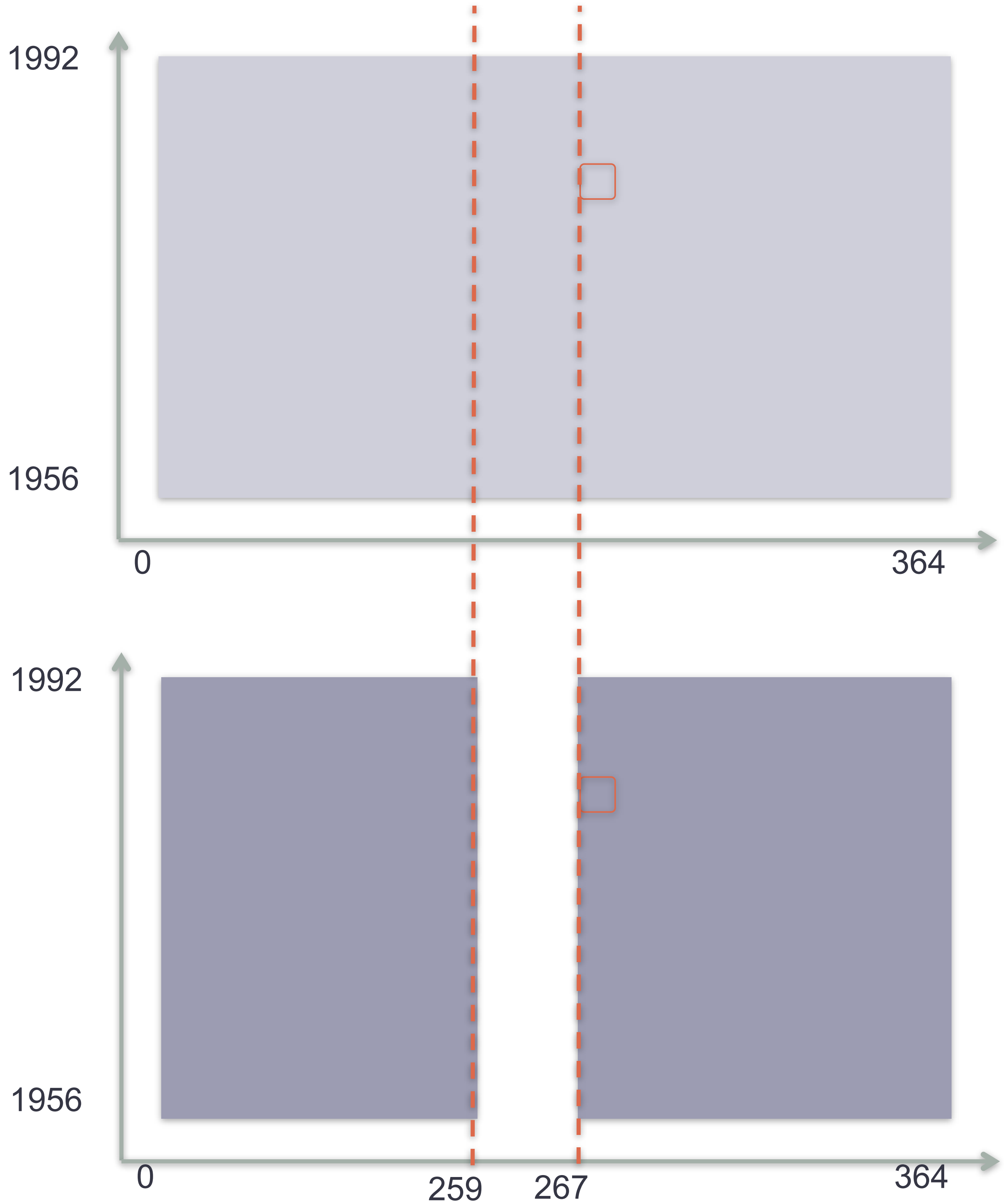
= $0 \le bday \le 364$

$1956 \le byear \le 1992$

each equally likely

} Assumption: this is accurate





bday-query1

today := 260;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0

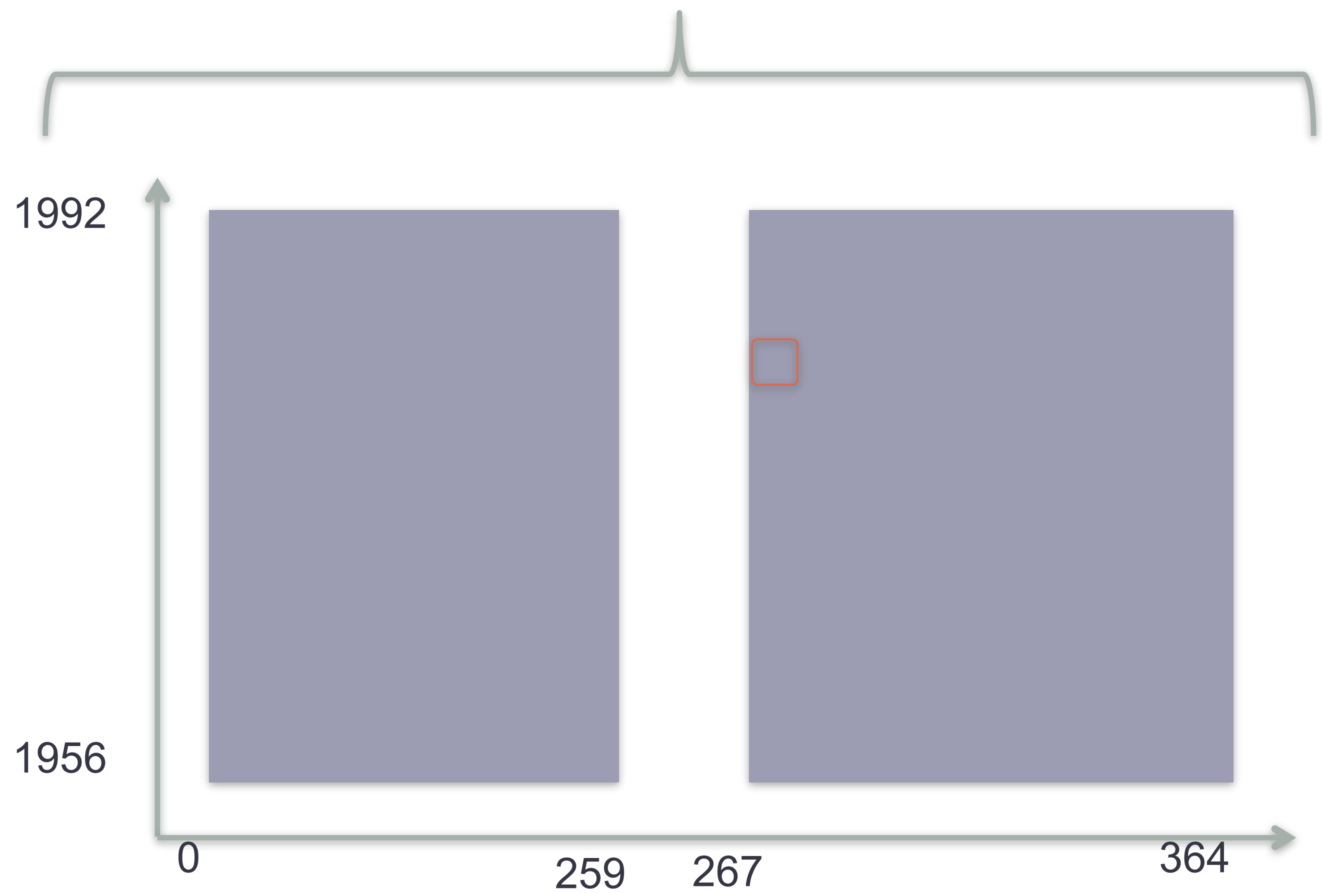
=  | (out = 0)

bday-query1

```
today := 260;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0
```

Problem

Policy: Is this acceptable?



=  | (out = 0)

= 

Idea: policy as knowledge threshold

- Answer a query if, for querier's revised belief, Pr[my secret] < *t*
  - Call *t* the **knowledge threshold**

- Choice of *t* depends on the risk of revelation

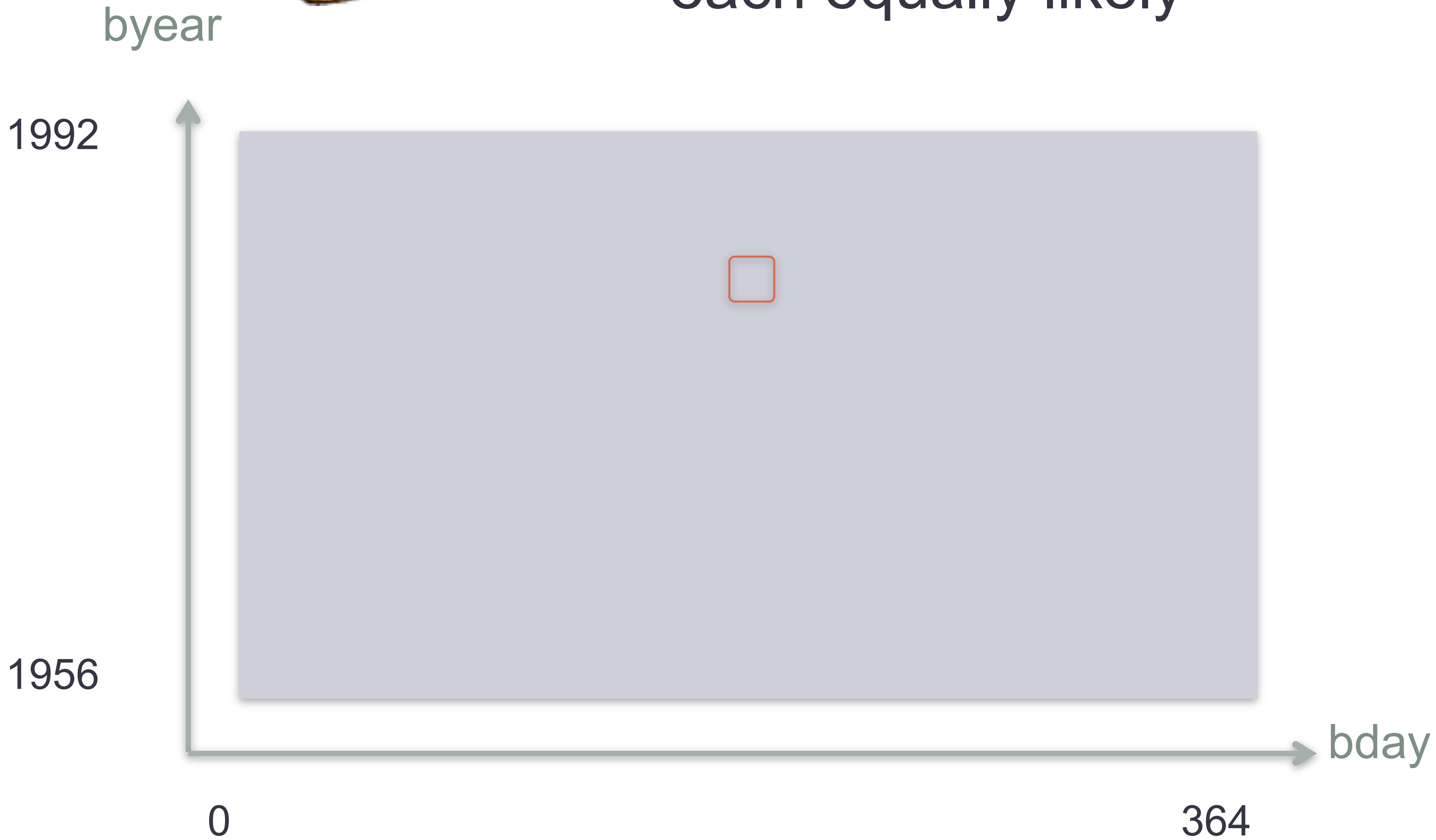
Bob’s policies

Bob (born September 24, 1980)
bday = 267
byear = 1980

} Secret

=

$0 \le \text{bday} \le 364$
$1956 \le \text{byear} \le 1992$
each equally likely

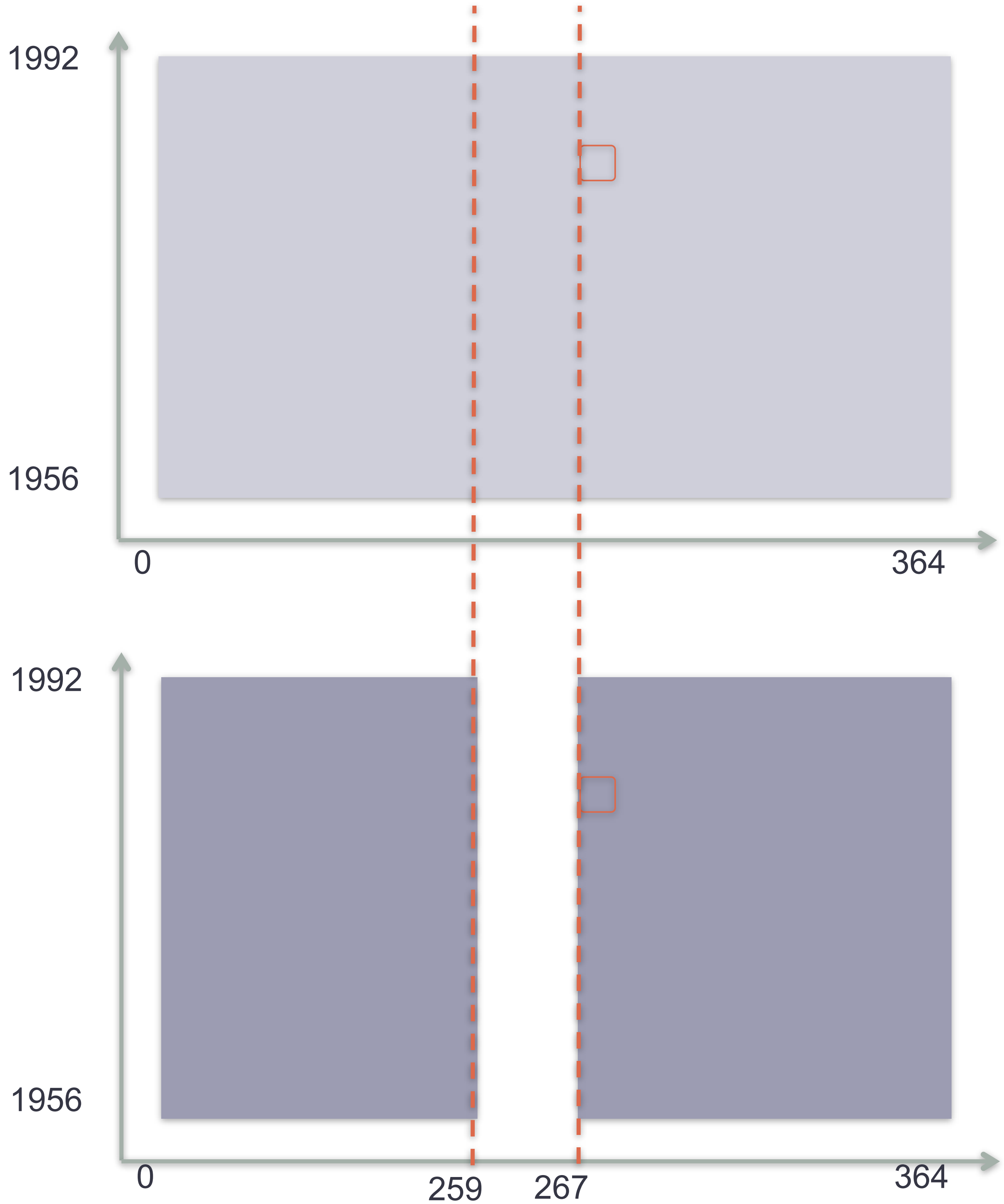


Pr[bday = 267] …

Policy
Pr[bday] < 0.2
Pr[bday,byear] < 0.05

Currently
Pr[bday] = 1/365
Pr[bday,byear] = 1/(365*37)

Back to the query ...



bday-query1

```
today := 260;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0
```

=  | (out = 0)

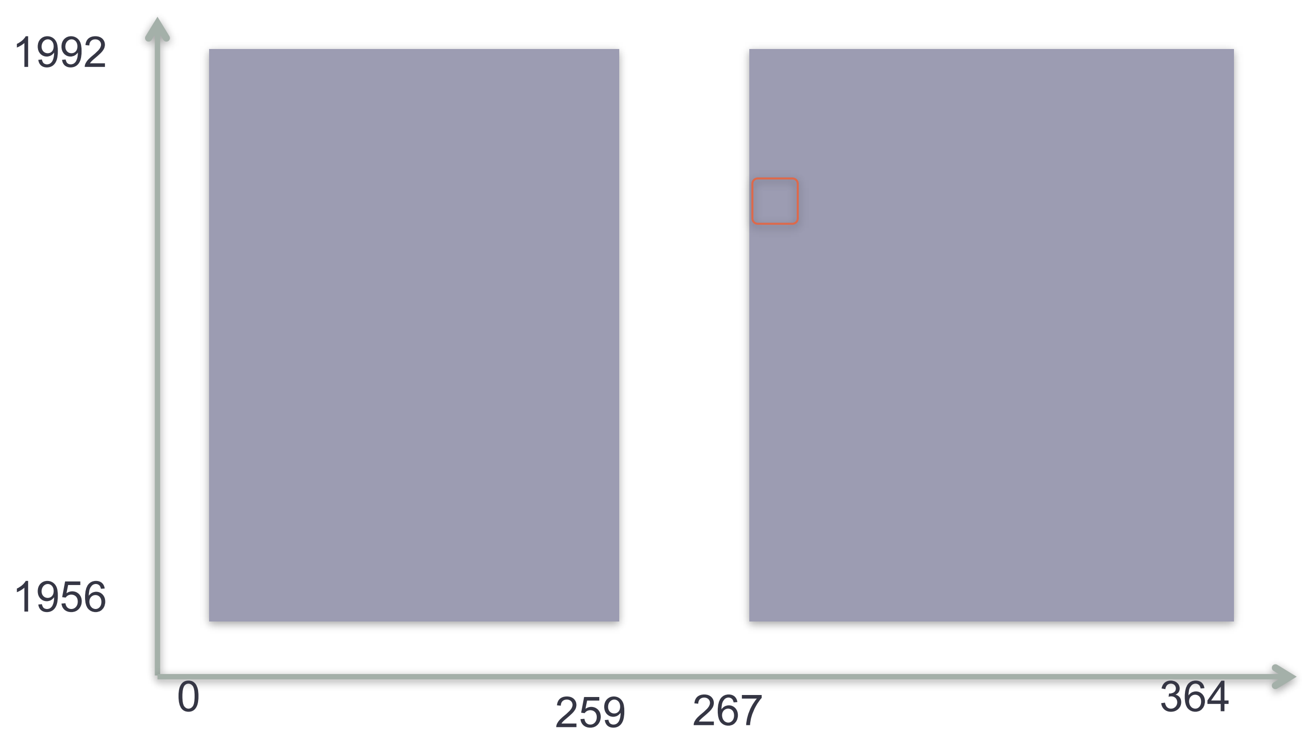
bday-query1

```
today := 260;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0
```

Potentially

Pr[bday] = 1/358 < 0.2

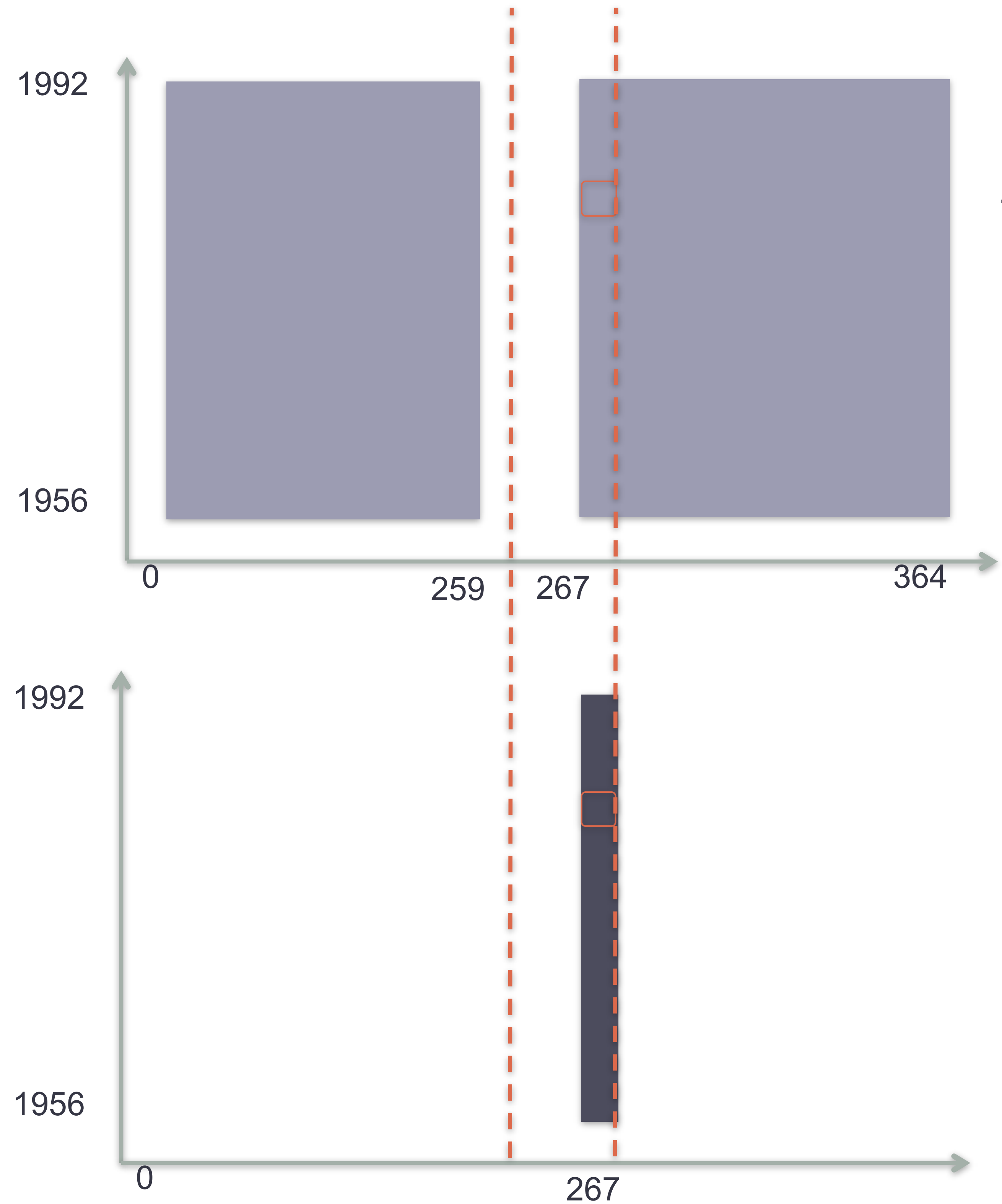
Pr[bday,byear] = 1/(358*37) < 0.05



=  | (out = 0)

= 

Next day …



bday-query2

today := **261**;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0

=  | (out = 1)

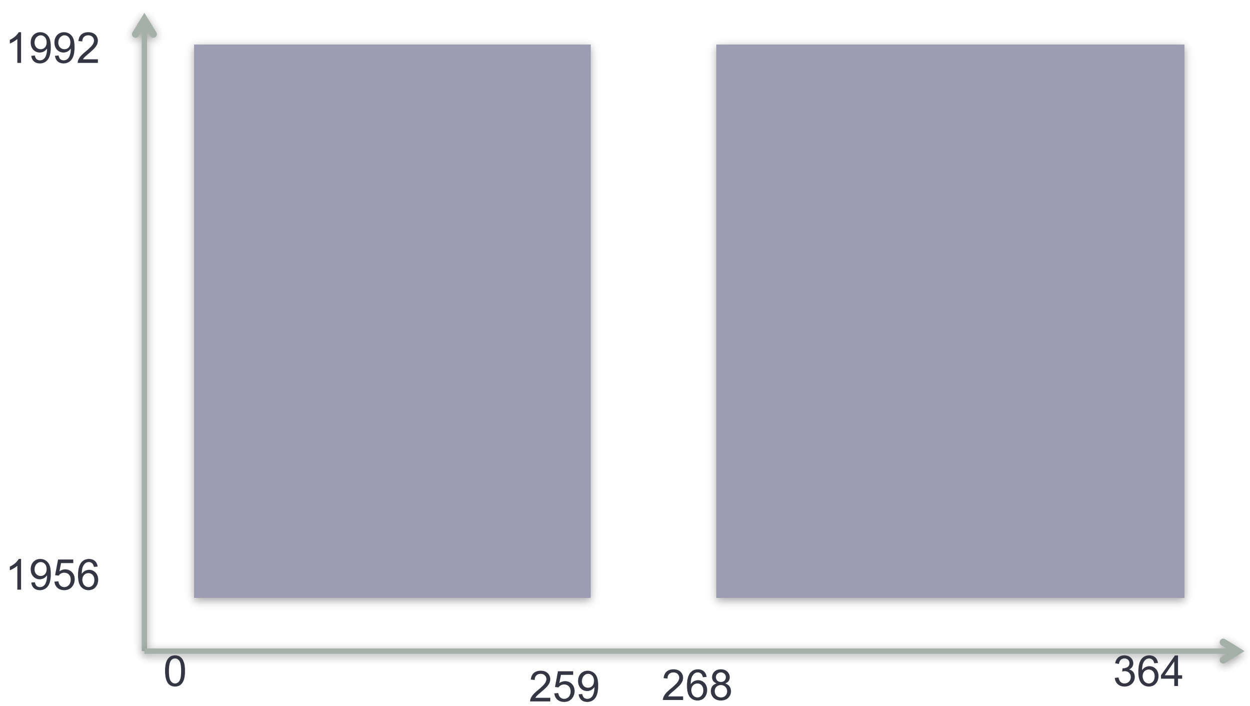
Pr[bday] = 1

So **reject**?

Querier's perspective

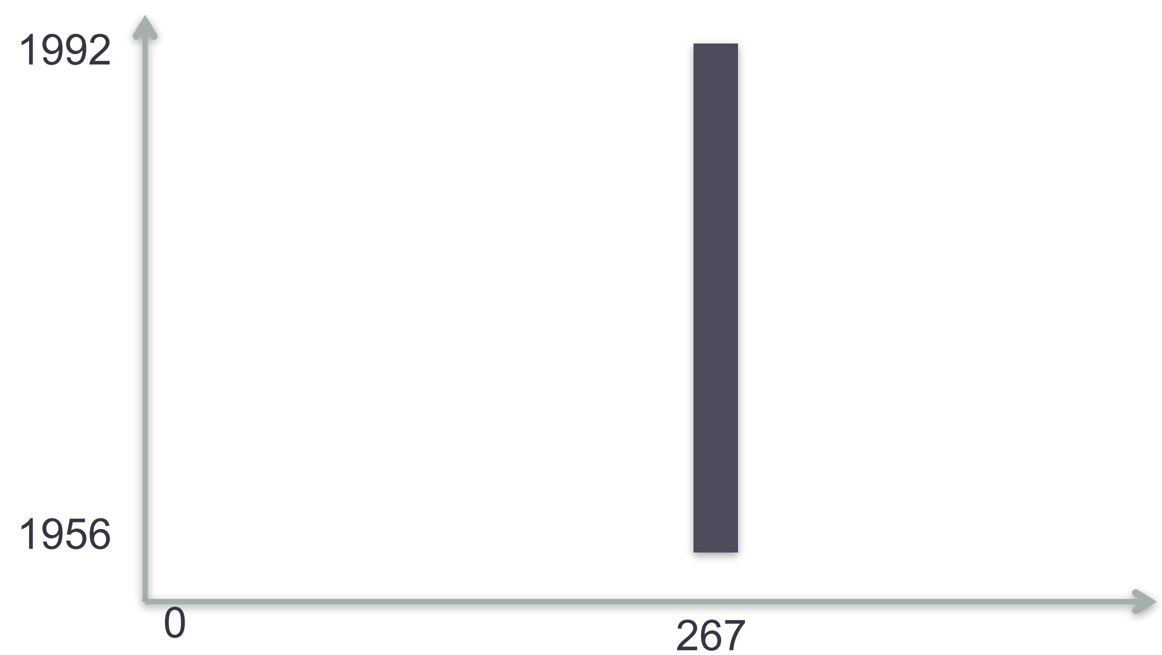
Assume querier knows policy

if bday ≠ 267



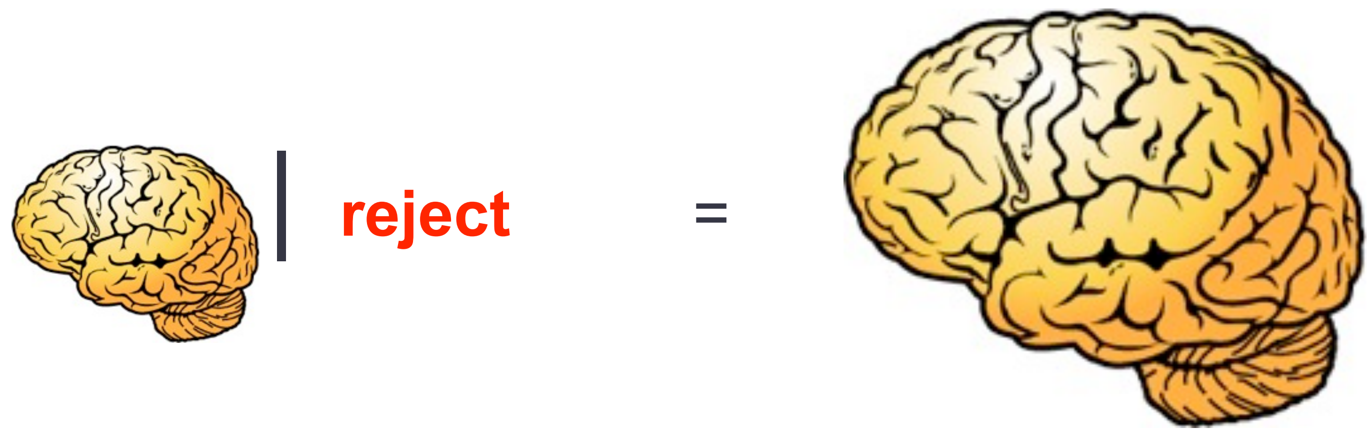
will get answer

if bday = 267



will get reject

Rejection problem



- Policy: Pr[bday = 267 | out = o] < *t*
  - Rejection, intended to protect secret, reveals secret!

Rejection revised

- Policy: Pr[bday = 267 | out = o] < *t*
- **Solution?**
  - Decide policy independently of secret
  - Revised policy
    - **for every possible output *o*,**
      - **for every possible bday *b*,**
        - **Pr[bday = b | out = o] < *t***
    - So the real *bday* in particular

bday-query1

```
today := 260;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0
```

accept

initial belief

cakes.com

bday-query2

today := **261**;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0

(after bday-query1)

reject

(regardless of what bday actually is)

(after bday-query1)

bday-query3

```
today := 266;
if bday ≤ today && bday < (today + 7)
  then out := 1
  else out := 0
```

accept

This is acceptable since it is five days after the last accept, keeping the probability within $t = 0.2$; i.e., $\Pr[266 \leq bday \leq 270] = 1/5$ if out =1, $\Pr[bday] = 1/353$ otherwise

Implementation

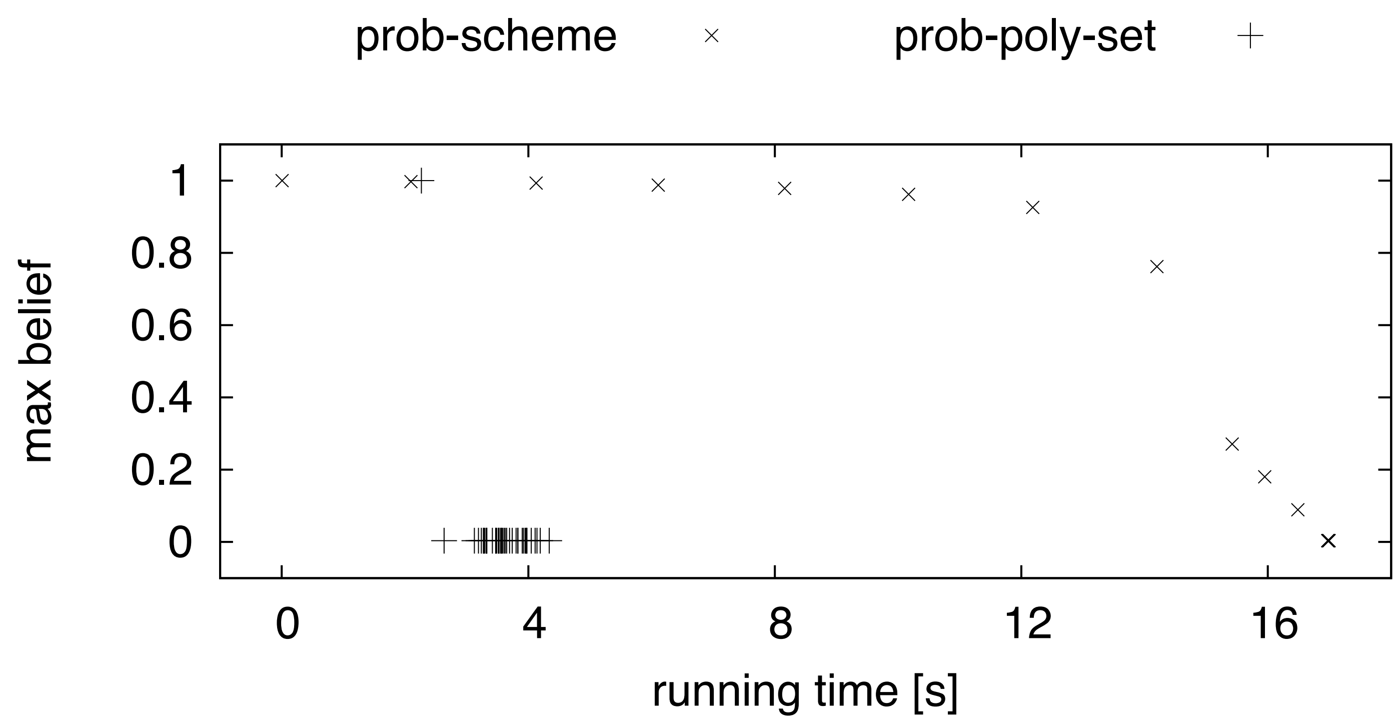
- Our query analysis in the style of abstract interpretation
  - We developed a novel probabilistic polyhedral domain
  - Scales far better than monte carlo sampling
- Precisely analyzes a particular sequence of queries, rather than all possible sequences
  - Far less conservative than considering all possible sequences of queries

Illustration of improved scalability

=

0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely

bday1 small



=

0 ≤ bday ≤ 364
1910 ≤ byear ≤ **2010**
each equally likely

bday 1 large

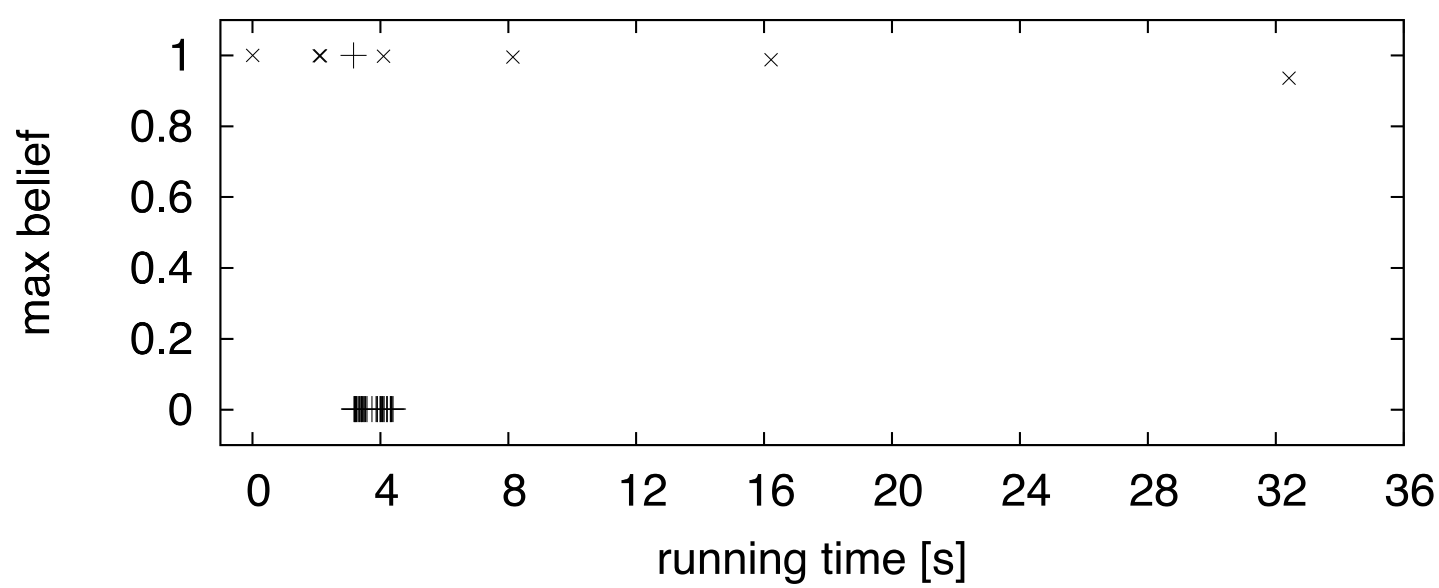
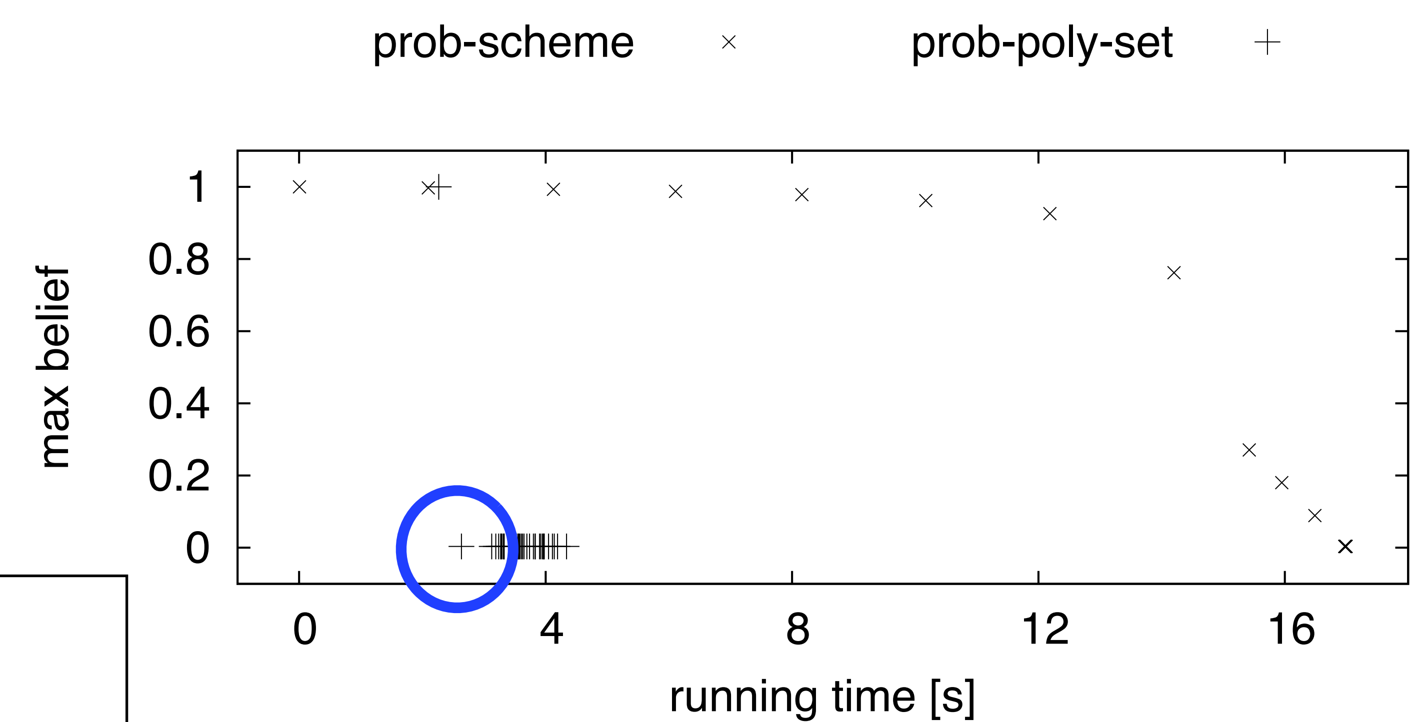


Illustration of improved scalability



 = 0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely

bday1 small

__Our approach__
best precision
time indep. of state size

 = 0 ≤ bday ≤ 364
1910 ≤ byear ≤ **2010**
each equally likely

bday 1 large

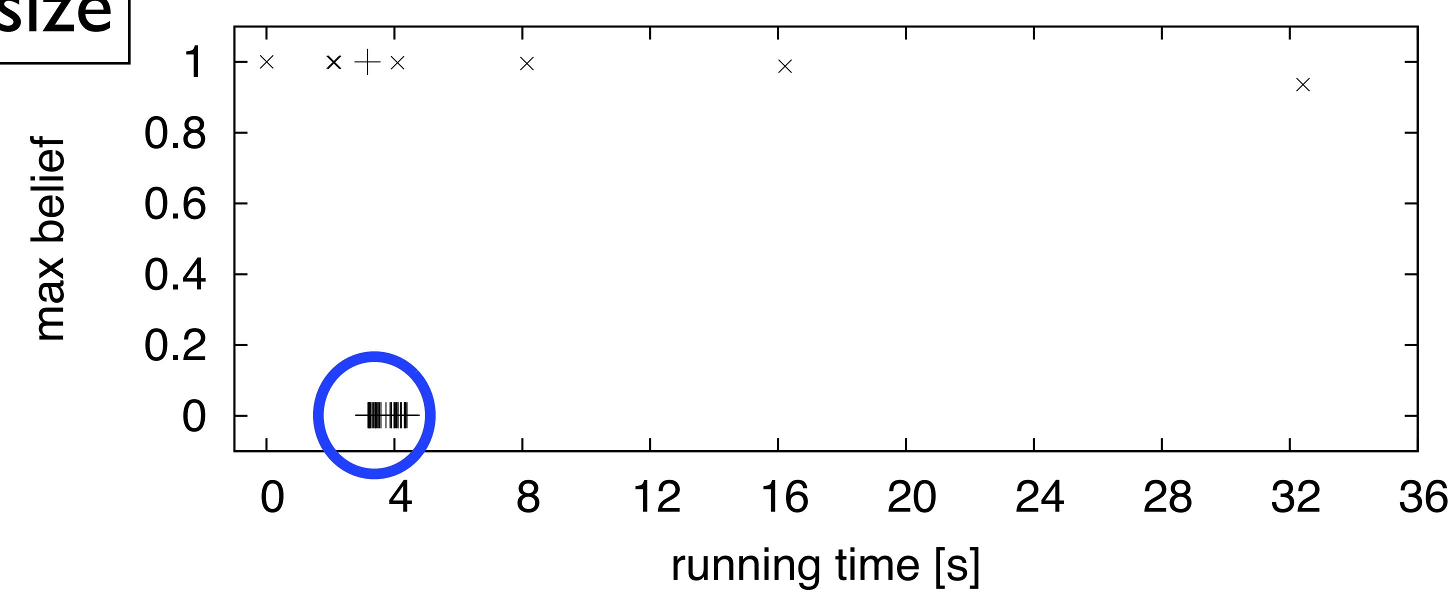


Illustration of improved scalability

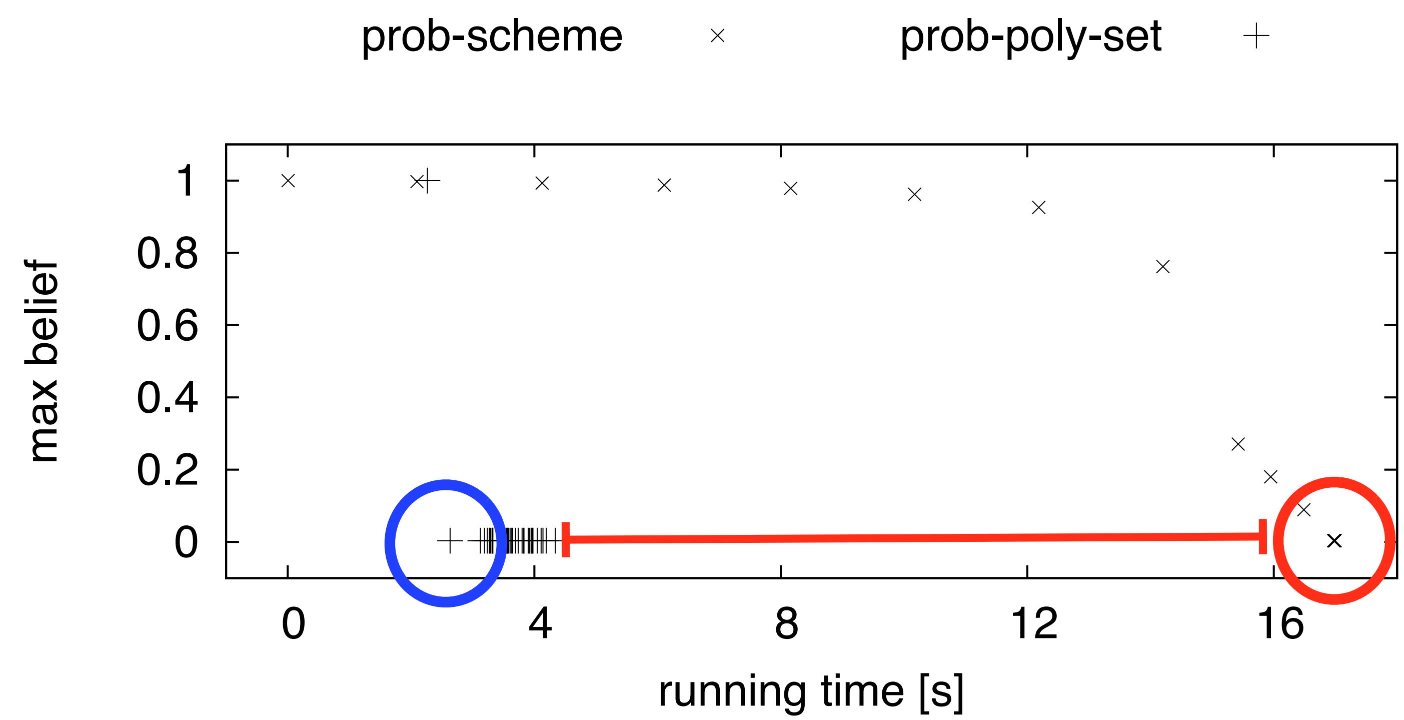
= 0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely

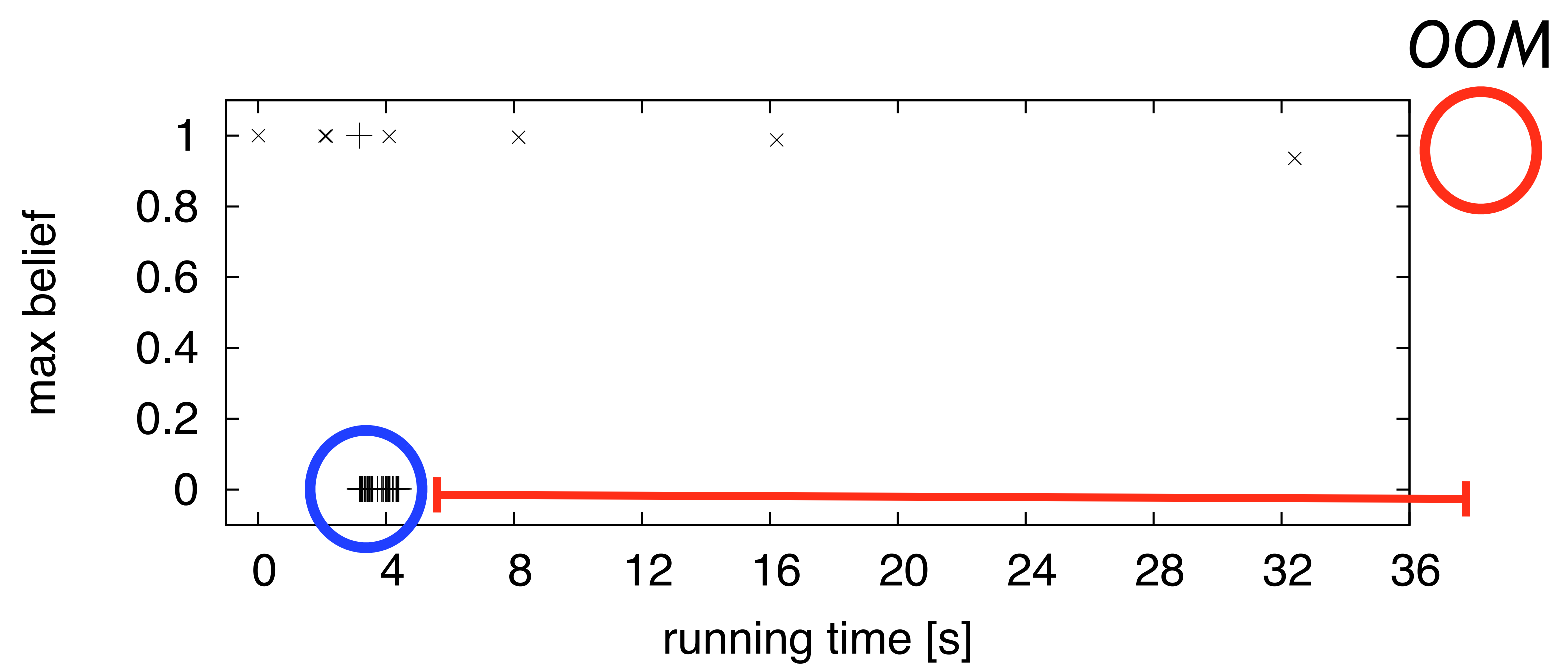
bday1 small

<u>Sampling</u>
same precision
much slower

= 0 ≤ bday ≤ 364
1910 ≤ byear ≤ **2010**
each equally likely

bday 1 large





Related work

- Significant work on database-oriented privacy, e.g., *differential privacy*. Key differences:
  - Trusts third party data provider to run safe aggregate queries. We work with individual data directly
  - DP's powerful adversary severely compromises utility, particularly for queries specific to individuals
  - Does not perform on-the-fly query analysis
- Also work on quantifying information flow
  - Tracks "bits leaked" but not relevant policies
  - Considers all possible query streams; too conservative

Current activities

- Application to *secure multiparty computation* [PLAS'12]
  - Two parties *p1*, *p2* have secrets *s1*, *s2* and compute compute *f(s1,s2)* = *x*, revealing only *x* to each
  - How much does *x* reveal about *s1* and *s2*?
- Time-indexed data: protect predictive features
  - Cooperative computations over coalition sensor networks
  - Ensuring anonymity of location traces [CCS'12]
- General direction: Privacy as a right

Collaborators (on analyses/tools)

- Former students / post-docs
  - Nikhil Swamy, Ph.D. 2008, @MSR Redmond
  - Polyvios Pratikakis, Ph.D. 2008, @FORTH Labs (Crete, Greece)
  - Avik Chaudhuri, post-doc 2009-10, @Adobe Research
  - Saurabh Srivastava, Ph.D. 2010, @Berkeley (CIFellow post-doc)
  - Martin Ma, Ph.D. 2011, @Amazon
  - Nataliya Guts, post-doc 2011-12, @Google
- Current students/post-docs
  - Khoo Yit Phang (7th year), Piotr Mardziel (4th year), Aseem Rastogi (4th year), Matt Hammer (post-doc)
- Profs/researchers
  - Jeff Foster (Maryland); Jonathan Katz (Maryland); Mudhakar Srivatsa (IBM T.J. Watson); Miguel Castro et al. (MSR Cambridge); Daan Leijen (MSR Redmond)

Other research

- Systems/networking research
  - Pavlos Papageorgiou (Ph.D, 2008), *Passive-Aggressive Measurement with MGRP* [SIGCOMM'09]
  - Justin McCann (Ph.D., 2012), *Automating Performance Diagnosis in Networked Systems*
- SCORE: Agile method for academic research [CACM'10]


SCORE: Agile Research Group Management | October 2010 | Communications of the ACM

http://cacm.acm.org/magazines/2010/10/99484-score-agile-research-group-management

TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS | ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

SIGN IN

COMMUNICATIONS OF THE ACM

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | MAGAZINE ARCH

Home / Magazine Archive / October 2010 (Vol. 53, No. 10) / SCORE: Agile Research Group Management / Full Text

VIEWPOINT

SCORE: Agile Research Group Management

By Michael Hicks, Jeffrey S. Foster
Communications of the ACM, Vol. 53 No. 10, Pages 30-31
10.1145/1831407.1831421
Comments (1)

VIEW AS: SHARE:

Credit: iStockPhoto.com

Working with and mentoring Ph.D. students is the central activity in running an academic research group. At the start of our careers as assistant professors, we took a fairly typical approach to managing student interactions: once or twice per week, we met with each of our students in prescheduled sessions of approximately half-hour or hour-long duration. However, this approach started breaking down as we gained more students and our other responsibilities increased: our time became fragmented and inefficiently used; hard-earned lessons were not shared effectively among students; and our group lacked any real cohesion or identity. In September 2006, we learned about Scrum,[1] an "agile" software development methodology, and realized we might be able to solve some of the problems we were having by adapting it to our research group.

In this Viewpoint, we briefly describe the resulting process, which we call SCORE (SCrum fOr REsearch). We have been using SCORE for several years, and have discovered it has many benefits, some we intended and some that surprised us. While every situation is different, we hope others may learn from our approach, in idea if not in form, and that we might inspire further discussion of research group management strategies. A longer version of this Viewpoint, with more information and space for feedback, is available at the SCORE Web page.[3]

Back to Top

SCORE

The major feature of SCORE is its meeting structure, which consists of two parts:

Regular all-hands status meetings. Several times a week (late mornings on Tuesdays, Wednesdays, and

SIGN IN for Full Access
User Name
Password
» Forgot Password?
» Create an ACM Web Account
SIGN IN

ARTICLE CONTENTS:
Introduction
SCORE
Why It Works for Us
Can SCORE Work for You?
References
Authors
Footnotes

MORE NEWS & OPINIONS
Intel Researchers Put Wi-Fi Inside—The Processor, That Is
ArsTechnica
'Girls Can't Program in Their Heads': Gender and Games in the Computing Classroom
Judy Robertson
Alan Turing's Other Universal Machine
Martin Campbell-Kelly

ACM RESOURCES

Maryland Cybersecurity Center (MC2)

- MC2 Director (since Oct 2011)
  - Two new CMSC faculty (Shi and Feamster)
  - Fifteen corporate partners (SAIC, NGC, Sourcefire, ...)
  - First MC2 Symposium, May 2011
  - Google Cybersecurity Seminars
  - ACES honors program, Prof. Masters, new courses
- Several new research initiatives underway
  - Privacy as a right
  - Anti-censorship

MC2 Maryland Cybersecurity Center

Summary: Building better software

- Along with colleagues and students, I am working to understand how to construct software that is available, reliable, and secure; i.e., software that
  - never crashes
  - adapts to changing circumstances and requirements
  - properly protects data
  - nevertheless provides useful and efficient services
- Programming languages, tools, and analyses, utilizing theory and implementation, are a powerful mechanism to this end

PL @ UM
PROGRAMMING LANGUAGES UNIVERSITY OF MARYLAND