Wysteria:
Secure Multiparty Computation via Functional Programming

Matthew A. Hammer
Aseem Rastogi, Michael Hicks

University of Maryland, Computer Science Dept.
December 2013

Outline

- Background:
  - Secure Multiparty Computation
- **Wysteria**
  - **Design elements via Examples**
  - Language Theory and Meta Theory
  - Implementation & Experimental Results

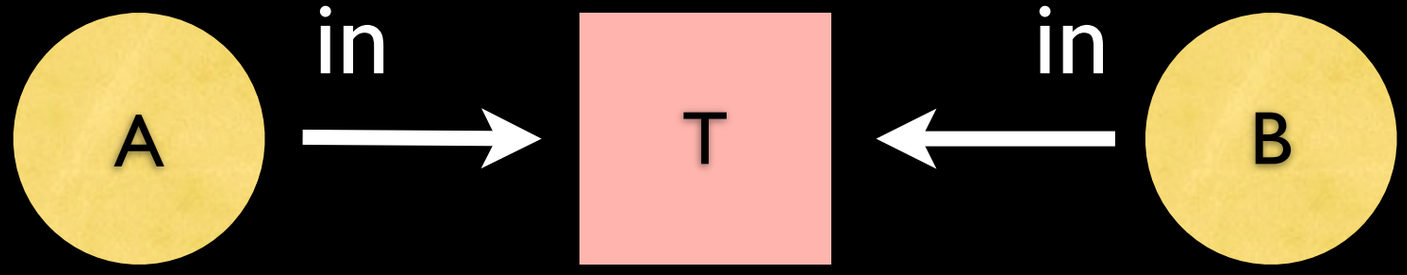
Background: Secure Multiparty Computation

Secure Multi-party Computation (SMC)

- SMC between A and B is **an abstraction**
- Simulates a **trusted third party** T
  - ***obliviously*** computes a public function *f*
  - input/output is private data of A and B

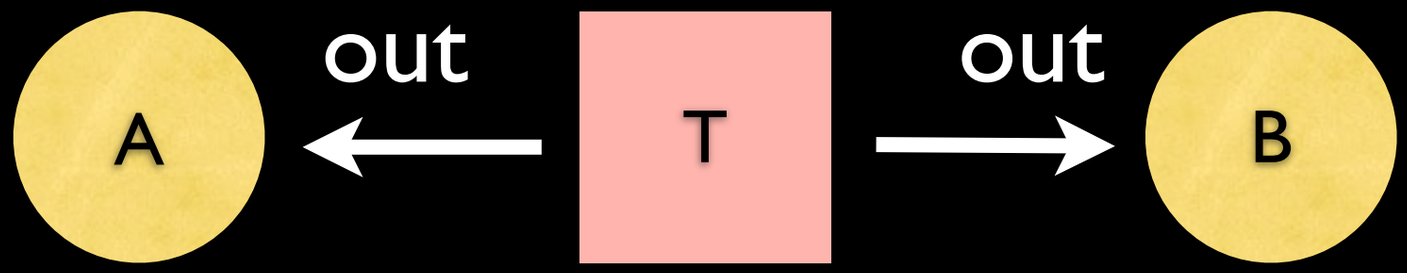
Secure Multi-party Computation (SMC)

1. Send inputs



2. Compute *f*

3. Recv outputs



Simple Examples of *f*

- Millionaire's problem
  - *Who is richer?*
- Private set intersection
  - *What elements are common?*
- Statistical calculations (e.g., median)
  - *Which element is ranked in the middle?*

Privacy Model for SMC

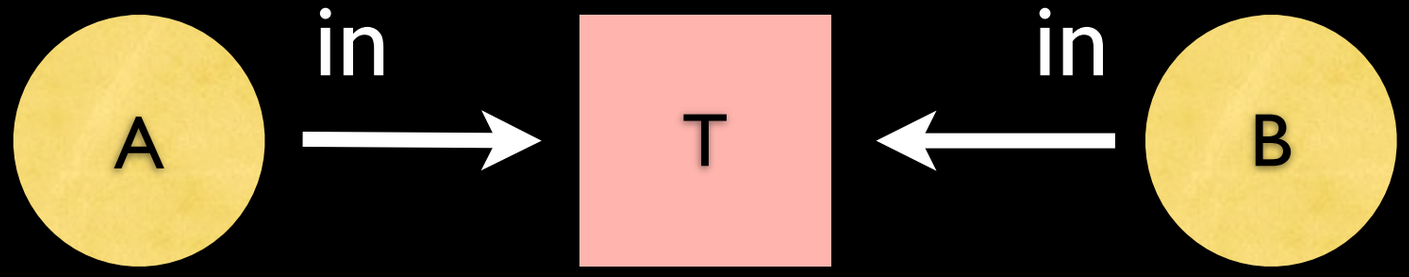
- Ideal functionality: All data computed is unknown*, up to knowledge implied by output of *f*
  - (*) computationally infeasible
  - Each output generally depends on all inputs
    - Information leakage is inherent to SMC
    - SMCs "declassify" private knowledge
  - **Design goal:** Protocols should be clear & concise!

Example of SMC in Practice

- Danish Beet Auction (2008--?)
  - Three servers: Danisco, DKS, SIMAP
  - 1200 bidders
  - 30 min to compute market clearing price for beets, via a double auction

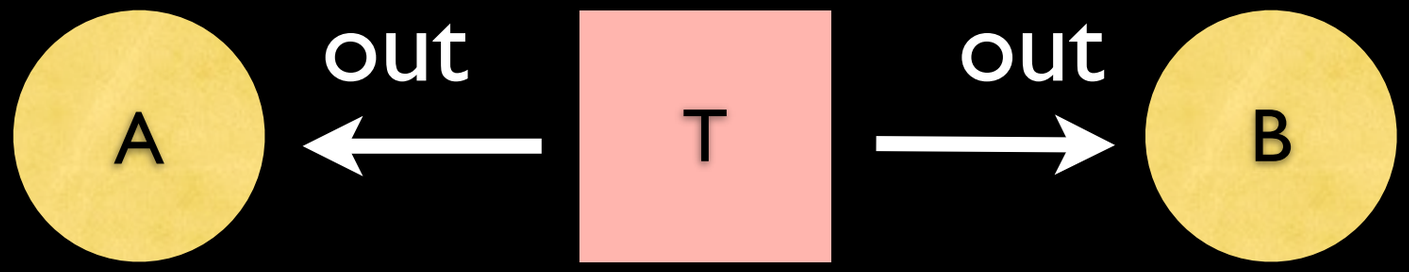
Mixing SMC with Local Computation

1. Send inputs



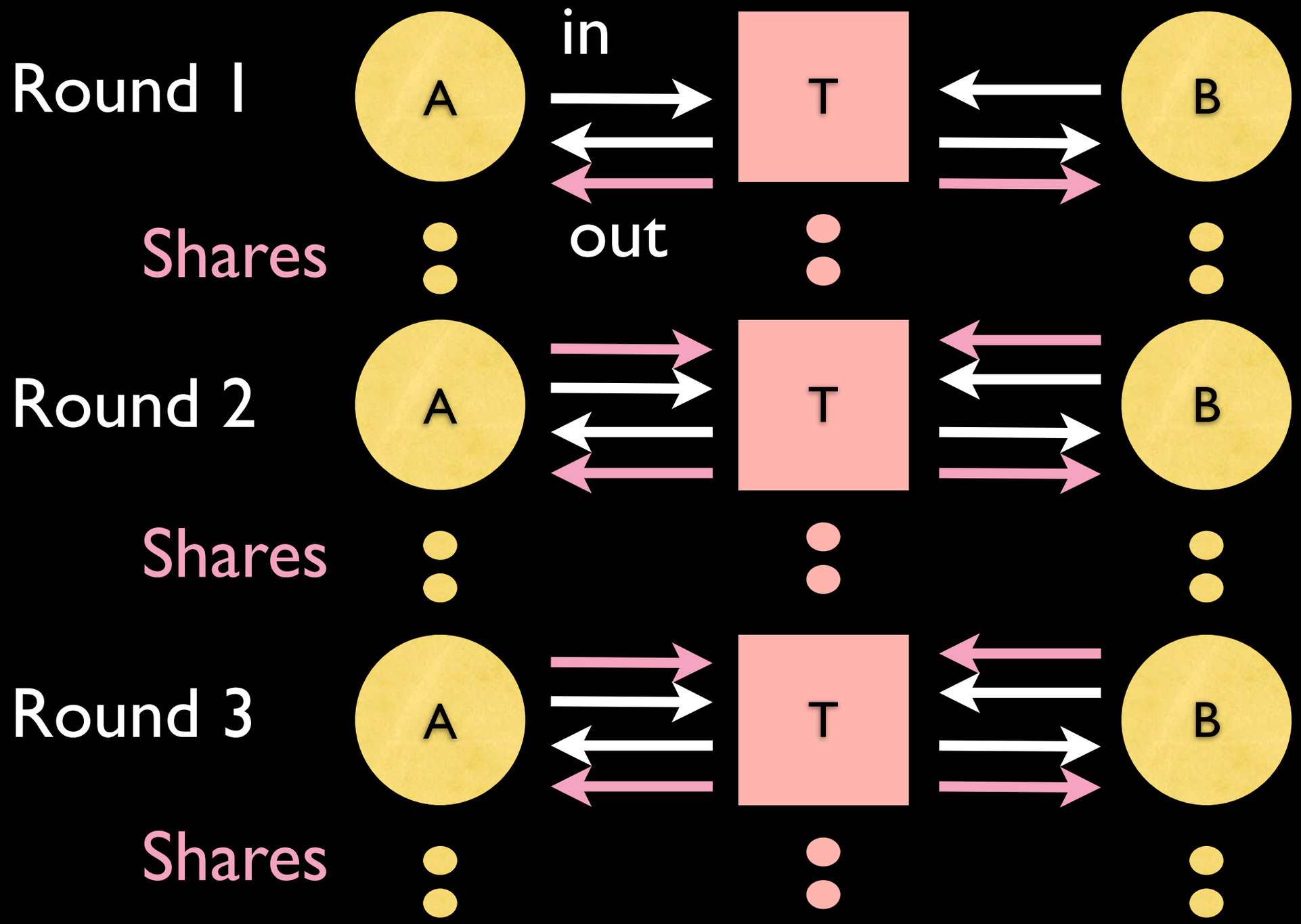
2. Compute *f*

3. Recv outputs



What about multiple interactions with T ?

Reactive SMC



(Potential) Reactive Examples

- **Card games** with betting (poker, bridge, etc.)
  - SMC used to *deal random cards*
  - Secure state consists of *card piles, private hands*
  - Shares force *commitment to private decisions*
- **Strategy games** with shared, private maps:
  - SMC used to *roll dice and simulate physics*
  - Secure state consists of a *shared map*
  - Shares enforce *coherence of private maps*

Strategy games

Shared Map

(partially unknown)

Detailed view

Related Work

- Fairplay [USENIX 2004]
- FairplayMP [CCS 2008]
- SMCL [PLAS 2007]
- TASTY [CCS 2010]
- L1 [COMPSAC 2011]
- PCF [USENIX 2013]
- Jif/Split [~2002]

No functional languages!

Library reuse?

Compositionality?

No formal semantics!

How to prove Correctness?

Wysteria: Design Elements and Examples

Wysteria

- Wysteria is an experimental PL design
- Design goal:
  - Compositional PL abstractions for generic, mixed-mode SMC protocols
- Design formula:
  - Simply-Typed Lambda Calculus + ???

Example: Millionaire's Problem

```
let a   =par(Alice)=          read () in
let b   =par(Bob)=            read () in
let out =sec({Alice,Bob})= ( a > b ) in
print out
```

Example: Millionaire's Problem

```
let a   =par(Alice)=          read () in
let b   =par(Bob)=            read () in
let out =sec({Alice,Bob})= ( a > b ) in
print out
```

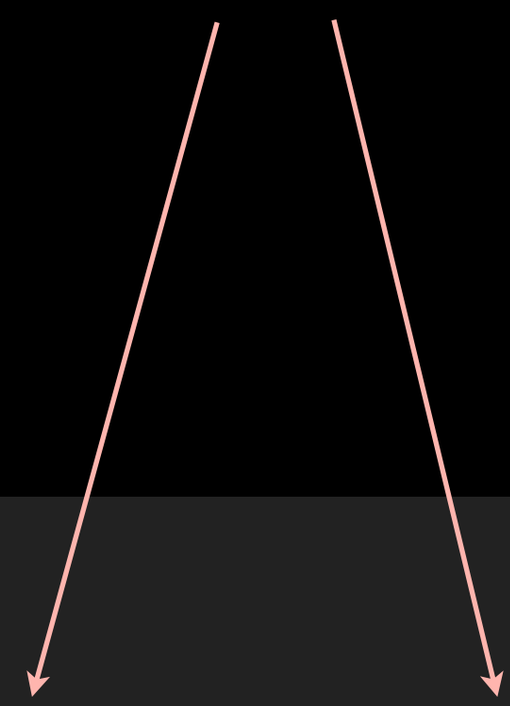
This is mixed mode

Not yet generic

Abstracting the Millionaire's Problem

```
is_richer =
  λx:W Alice int.
  λy:W Bob int.
    let out =sec({Alice,Bob})=
       x[Alice] > y[Bob]
    in out
```

Function abstracts over private inputs via wire bundles



```
let a   =par(Alice)=          read () in
let b   =par(Bob)=            read () in
let out =par(Alice,Bob)= is_richer (wire Alice a) (wire Bob b)
in print out
```

Abstracting the Millionaire's Problem

```
is_richer =
  λx:W {Alice,Bob} int.
    let out =sec({Alice,Bob})=
       x[Alice] > x[Bob]
    in out
```

wire bundles concatenate

```
let a   =par(Alice)=        read () in
let b   =par(Bob)=          read () in
let out =par(Alice,Bob)=
        is_richer ((wire Alice a) ++ (wire Bob b))
in print out
```

Abstracting the Millionaire's Problem

Generic protocol:
Abstracts over variable principal set

```
richest_of = λms : ps.  λx :W ms int.
    let out =sec(ms)=
    wfold None x
    λrichest. λp. λn.  match richest with
      | None ⇒ Some p
      | Some q ⇒ if n > x[q] then Some p else Some q
  in out
```

Abstracting the Millionaire's Problem

```
let all = {A, B} in
let input = (wire A a) ++ (wire B b) in
let r : ps{singl ∧ ⊆all} option = (richest_of all) input in
```

```
richest_of = λms : ps.  λx :W ms int.
let out =sec(ms)=
  wfold None x
  λrichest. λp. λn.  match richest with
    | None ⇒ Some p
    | Some q ⇒ if n > x[q] then Some p else Some q
in out
```

Abstracting the Millionaire's Problem

```
let all = {A, B, C} in
let input = (wire A a) ++ (wire B b) ++ (wire C c) in
let r : ps{singl ∧ ⊆all} option = (richest_of all) input in
```

```
richest_of = λms : ps.  λx :W ms int.
let out =sec(ms)=
  wfold None x
  λrichest. λp. λn.  match richest with
    | None ⇒ Some p
    | Some q ⇒ if n > x[q] then Some p else Some q
in out
```

Two-round Betting Game (1/2)

Round 1

```
let a1 =par(A)= read () in
let b1 =par(B)= read () in
let in1 = (wire A a1) ++ (wire B b1) in
let (higher1, sa, sb) =sec(A,B)=
  let c = if in1[A] > in2[B] then A else B in
  (c,  makesh in1[A],  makesh in2[B])
in print higher1
...
```

Two-round Betting Game (2/2)

Round 2

```
• • •
let a2 =par(A)= read () in
let b2 =par(B)= read () in
let in2 = (wire A a2) ++ (wire B b2) in
let higher2 =sec(A,B)=
  let (a1, b1) = (combsh sa, combsh sb) in
  let bida = (a1 + in2[A]) / 2 in
  let bidb = (b1 + in2[B]) / 2 in
  let c = if bida > bidb then A else B in
in print higher2
```

Wysteria's Design Elements

- **Principals as data**
  - Principals can be inputs and outputs of computation
- **Mixed-mode** computation
  - Designates: Secure vs parallel
  - Designates: Participants
- **Wire bundles** of private data
- **Shares** of secure state

Wysteria Language: Theory & Meta Theory

Wysteria Language Theory

- **Type system** reasons about special abstractions
  - (i.e., principals, wire bundles, modal computation, shares, etc.)

- (Two) Operational semantics:
  - **Single-threaded view** for synchrony
  - **Multi-threaded view** for privacy + parallelism

Γ ⊢M e : τ ; ε

M ::= sec(w) | par(w)

Type System

$\boxed{\Gamma \vdash_M v : \tau}$ *(Value typing)*

T-VAR
$$\frac{x :_M \tau \in \Gamma \quad \Gamma \vdash_M \tau}{\Gamma \vdash_M x : \tau}$$

T-UNIT
$$\frac{}{\Gamma \vdash_M () : \mathbf{unit}}$$

T-INJ
$$\frac{\Gamma \vdash_M v : \tau_i \quad j \in \{1,2\} \quad \tau_j\ \texttt{IsFlat}}{\Gamma \vdash_M \mathbf{inj}_i\, v : \tau_1 + \tau_2}$$

T-PROD
$$\frac{\Gamma \vdash_M v_i : \tau_i}{\Gamma \vdash_M (v_1, v_2) : \tau_1 \times \tau_2}$$

T-PRINC
$$\frac{}{\Gamma \vdash_M p : \mathbf{ps}\ (\nu = \{p\})}$$

T-PSONE
$$\frac{\Gamma \vdash_M w : \mathbf{ps}\ (\mathbf{singl}(\nu))}{\Gamma \vdash_M \{w\} : \mathbf{ps}\ (\nu = \{w\})}$$

T-PSUNION
$$\frac{\Gamma \vdash_M w_i : \mathbf{ps}\ \phi_i}{\Gamma \vdash_M w_1 \cup w_2 : \mathbf{ps}\ (\nu = w_1 \cup w_2)}$$

T-PSVAR
$$\frac{\Gamma \vdash_M x : \mathbf{ps}\ \phi}{\Gamma \vdash_M x : \mathbf{ps}\ (\nu = x)}$$

T-MSUB
$$\frac{\Gamma \vdash_M x : \tau \quad \Gamma \vdash M \rhd N \quad N = \mathsf{s}(\_) \Rightarrow \tau\ \texttt{IsSecIn}}{\Gamma \vdash_N x : \tau}$$

T-SUB
$$\frac{\Gamma \vdash_M v : \tau_1 \quad \Gamma \vdash_M \tau_1 <: \tau}{\Gamma \vdash_M v : \tau}$$

Fig. 2. Value typing judgement.

$\boxed{\Gamma \vdash_M \tau_1 <: \tau_2}$ *(Subtyping)*

S-REFL
$$\frac{}{\Gamma \vdash_M \tau <: \tau}$$

S-TRANS
$$\frac{\Gamma \vdash_M \tau_1 <: \tau_2 \quad \Gamma \vdash_M \tau_2 <: \tau_3}{\Gamma \vdash_M \tau_1 <: \tau_3}$$

S-SUM
$$\frac{\Gamma \vdash_M \tau_i <: \tau_i'}{\Gamma \vdash_M \tau_1 + \tau_2 <: \tau_1' + \tau_2'}$$

S-PROD
$$\frac{\Gamma \vdash_M \tau_i <: \tau_i'}{\Gamma \vdash_M \tau_1 \times \tau_2 <: \tau_1' \times \tau_2'}$$

S-PRINCS
$$\frac{[\![\Gamma]\!] \vDash \phi_1 \Rightarrow \phi_2}{\Gamma \vdash_M \mathbf{ps}\ \phi_1 <: \mathbf{ps}\ \phi_2}$$

S-WIRE
$$\frac{\Gamma \vdash_M w_2 : \mathbf{ps}\ (\nu \subseteq w_1) \quad \Gamma \vdash_M \tau_1 <: \tau_2}{\Gamma \vdash_M \mathbf{W}\, w_1\, \tau_1 <: \mathbf{W}\, w_2\, \tau_2}$$

S-ARRAY
$$\frac{\Gamma \vdash_M \tau_1 <: \tau_2 \quad \Gamma \vdash_M \tau_2 <: \tau_1}{\Gamma \vdash_M \mathbf{Array}\, \tau_1 <: \mathbf{Array}\, \tau_2}$$

S-SHARE
$$\frac{\Gamma \vdash_M w_2 : \mathbf{ps}\ (\nu = w_1) \quad \Gamma \vdash_M \tau_1 <: \tau_2 \quad \Gamma \vdash_M \tau_2 <: \tau_1}{\Gamma \vdash_M \mathbf{Sh}\, w_1\, \tau_1 <: \mathbf{Sh}\, w_2\, \tau_2}$$

S-ARROW
$$\frac{\Gamma \vdash_M \tau_1' <: \tau_1 \quad \Gamma \vdash_M \tau_2 <: \tau_2'}{\Gamma \vdash_M \tau_1 \xrightarrow{\epsilon} \tau_2 <: \tau_1' \xrightarrow{\epsilon} \tau_2'}$$

$\boxed{\Gamma \vdash M \rhd N}$ *(Mode M can delegate to mode N)*

D-REFL
$$\frac{}{\Gamma \vdash M \rhd M}$$

D-TOP
$$\frac{}{\Gamma \vdash \top \rhd M}$$

D-PAR
$$\frac{\Gamma \vdash_{\mathsf{p}(w_1)} w_2 : \mathbf{ps}\ (\nu \subseteq w_1)}{\Gamma \vdash \mathsf{p}(w_1) \rhd \mathsf{p}(w_2)}$$

D-SEC
$$\frac{\Gamma \vdash_{\mathsf{p}(w_1)} w_2 : \mathbf{ps}\ (\nu = w_1)}{\Gamma \vdash \mathsf{p}(w_1) \rhd \mathsf{s}(w_2)}$$

Fig. 3. Subtyping and delegation judgements.

$\boxed{\Gamma \vdash_M e : \tau ; \epsilon}$

*(**Expression typing**: "Under Γ, expression e has type τ, and may be run at M. ")*

T-FST
$$\frac{\Gamma \vdash_M v : \tau_1 \times \tau_2}{\Gamma \vdash_M \mathbf{fst}\,(v) : \tau_1 ; .}$$

T-SND
$$\frac{\Gamma \vdash_M v : \tau_1 \times \tau_2}{\Gamma \vdash_M \mathbf{snd}\,(v) : \tau_2 ; .}$$

T-CASE
$$\frac{(M = \mathsf{p}(\_) \wedge \epsilon = M) \vee (\tau\ \texttt{IsFO}) \quad \Gamma \vdash_M v : \tau_1 + \tau_2 \quad \Gamma, x_i :_M \tau_i \vdash_M e_i : \tau ; \epsilon_i}{\Gamma \vdash_M \mathbf{case}\,(v, x_1.e_1, x_2.e_2) : \tau ; \epsilon, \epsilon_1, \epsilon_2}$$

T-APP
$$\frac{M = \mathsf{s}(\_) \Rightarrow \tau\ \texttt{IsFO} \quad \Gamma \vdash^{\mathsf{app}}_M \tilde{e} : \tau ; \epsilon}{\Gamma \vdash_M \tilde{e} : \tau ; \epsilon}$$

T-LET
$$\frac{M = m(\_) \quad N = \_(w) \quad \Gamma \vdash M \rhd N \quad \Gamma \vdash_N e_1 : \tau_1 ; \epsilon_1 \quad \Gamma, x :_{m(w)} \tau_1 \vdash_M e_2 : \tau_2 ; \epsilon_2 \quad \Gamma \vdash_M \tau_2}{\Gamma \vdash_M \mathbf{let}\ x \overset{N}{=} e_1\ \mathbf{in}\ e_2 : \tau_2 ; N, \epsilon_1, [\![\epsilon_2]\!]^{\Gamma}_M}$$

T-FIX
$$\frac{M = \mathsf{p}(\_) \quad \Gamma \vdash M \rhd \epsilon \quad \Gamma, x :_M (y{:}\tau_1 \xrightarrow{\epsilon} \tau_2) \vdash_M e : (y{:}\tau_1 \xrightarrow{\epsilon} \tau_2) ; \epsilon}{\Gamma \vdash_M \mathbf{fix}\, x : (y{:}\tau_1 \xrightarrow{\epsilon} \tau_2).e : (y{:}\tau_1 \xrightarrow{\epsilon} \tau_2) ; \epsilon, M}$$

T-ARRAY
$$\frac{M = \mathsf{p}(\_) \quad \Gamma \vdash_M v_1 : \mathbf{nat} \quad \Gamma \vdash_M v_2 : \tau}{\Gamma \vdash_M \mathbf{array}\, v_1\, v_2 : \mathbf{Array}\, \tau ; .}$$

T-SELECT
$$\frac{\Gamma \vdash_M v_1 : \mathbf{Array}\, \tau \quad \Gamma \vdash_M v_2 : \mathbf{nat}}{\Gamma \vdash_M \mathbf{select}\, v_1\, v_2 : \tau ; .}$$

T-UPDATE
$$\frac{M = \mathsf{p}(\_) \quad \mathsf{mode}(v_1, \Gamma) = M \quad \Gamma \vdash_M v_1 : \mathbf{Array}\, \tau \quad \Gamma \vdash_M v_2 : \mathbf{nat} \quad \Gamma \vdash_M v_3 : \tau}{\Gamma \vdash_M \mathbf{update}\, v_1\, v_2\, v_3 : \mathbf{unit} ; .}$$

T-WIRE
$$\frac{m = \mathsf{s} \Rightarrow \tau\ \texttt{IsFO} \quad \tau\ \texttt{IsFlat} \quad \Gamma \vdash_{\mathsf{p}(w_2)} w_1 : \mathbf{ps}\ (\nu \subseteq w_2) \quad \Gamma \vdash_{m(w_1)} v : \tau}{\Gamma \vdash_{m(w_2)} \mathbf{wire}_{w_1}(v) : \mathbf{W}\, w_1\, \tau ; .}$$

T-WPROJ
$$\frac{m = \mathsf{p} \Rightarrow \phi = (\nu = w_1) \quad m = \mathsf{s} \Rightarrow \phi = (\nu \subseteq w_1) \quad \Gamma \vdash_{m(w_1)} v : \mathbf{W}\, w_2\, \tau \quad \Gamma \vdash_{\mathsf{p}(w_1)} w_2 : \mathbf{ps}\ (\phi \wedge \mathbf{singl}(\nu))}{\Gamma \vdash_{m(w_1)} v[w_2] : \tau ; .}$$

T-WIREUN
$$\frac{\Gamma \vdash_M v_1 : \mathbf{W}\, w_1\, \tau \quad \Gamma \vdash_M v_2 : \mathbf{W}\, w_2\, \tau}{\Gamma \vdash_M v_1 \mathbin{+\!\!+} v_2 : \mathbf{W}\, (w_1 \cup w_2)\, \tau ; .}$$

T-WFOLD
$$\frac{M = \mathsf{s}(\_) \quad \tau_2\ \texttt{IsFO} \quad \phi = (\nu \subseteq w \wedge \mathbf{singl}(\nu)) \quad \Gamma \vdash_M v_1 : \mathbf{W}\, w\, \tau \quad \Gamma \vdash_M v_2 : \tau_2 \quad \Gamma \vdash^{\mathsf{app}}_M \tilde{e} : \tau_2 \dot{\to} \mathbf{ps}\ \phi \dot{\to} \tau \dot{\to} \tau_2 ; .}{\Gamma \vdash_M \mathbf{wfold}_w(v_1, v_2, \tilde{e}) : \tau_2 ; .}$$

T-WAPP
$$\frac{M = \mathsf{p}(\_) \quad \Gamma \vdash_M v_1 : \mathbf{W}\, w\, \tau_1 \quad \Gamma \vdash_M v_2 : \mathbf{W}\, w\, (\tau_1 \dot{\to} \tau_2)}{\Gamma \vdash_M \mathbf{wapp}_w(v_1, v_2) : \mathbf{W}\, w\, \tau_2 ; .}$$

T-WAPS
$$\frac{M = \mathsf{s}(\_) \quad \tau_2\ \texttt{IsFO} \quad \Gamma \vdash_M v_1 : \mathbf{W}\, w\, \tau_1 \quad \Gamma \vdash_M v_2 : \tau_1 \dot{\to} \tau_2}{\Gamma \vdash_M \mathbf{waps}_w(v_1, v_2) : \mathbf{W}\, w\, \tau_2 ; .}$$

T-WCOPY
$$\frac{M = \mathsf{p}(w_1) \quad \Gamma \vdash_M w_2 : \mathbf{ps}\ (\nu \subseteq w_1) \quad \Gamma \vdash_{\mathsf{p}(w_2)} v : \mathbf{W}\, w_2\, \tau}{\Gamma \vdash_M \mathbf{wcopy}_{w_2} v : \mathbf{W}\, w_2\, \tau ; .}$$

T-MAKESH
$$\frac{M = \mathsf{s}(w) \quad \tau\ \texttt{IsFO} \quad \tau\ \texttt{IsFlat} \quad \Gamma \vdash_M v : \tau}{\Gamma \vdash_M \mathbf{makesh}\, v : \mathbf{Sh}\, w\, \tau ; .}$$

T-COMBSH
$$\frac{M = \mathsf{s}(w) \quad \Gamma \vdash_M v : \mathbf{Sh}\, w\, \tau}{\Gamma \vdash_M \mathbf{combsh}\, v : \tau ; .}$$

$\boxed{\Gamma \vdash^{\mathsf{app}}_M \tilde{e} : \tau ; \epsilon}$ *(Application typing)*

TA-LAM
$$\frac{\Gamma \vdash_M \tau \quad \Gamma, x :_M \tau \vdash_M e : \tau_1 ; \epsilon}{\Gamma \vdash^{\mathsf{app}}_M \lambda x : \tau.e : (x{:}\tau \xrightarrow{\epsilon} \tau_1) ; .}$$

TA-VAR
$$\frac{\Gamma \vdash_M x : \tau}{\Gamma \vdash^{\mathsf{app}}_M x : \tau ; .}$$

TA-APP
$$\frac{\Gamma \vdash^{\mathsf{app}}_M \tilde{e} : x{:}\tau_1 \xrightarrow{\epsilon} \tau_2 ; \epsilon_1 \quad \Gamma \vdash_M v : \tau_1 \quad \Gamma \vdash M \rhd \epsilon[v/x]}{\Gamma \vdash^{\mathsf{app}}_M \tilde{e}\, v : \tau_2[v/x] ; \epsilon_1, \epsilon[v/x]}$$

Fig. 4. Expression typing judgements.

Type System

$\Gamma \vdash_M e : \tau ; \epsilon$

M ::= sec(w) | par(w)

Values: Standard stuff + **Principal sets**

Standard stuff (datatypes and higher-order functions)

mode composition

parallel-mode loops

array (intro)

array read

array write

wire (intro)

Wire elim forms (projection, folding, mapping)

Shares (intro & elim rules)

Standard stuff (function application)

Sub-typing: **Refinement implication**

Delegation

Fig. 2. Value typing judgement.

Fig. 3. Subtyping and delegation judgements.

Fig. 4. Expression typing judgements.

Single-threaded Semantics

$C_1 \longrightarrow C_2$

M ::= sec(w) | par(w)
C ::= M{σ;κ;ψ;e}

store
stack
environment
program counter

$\boxed{\mathcal{C}_1 \longrightarrow \mathcal{C}_2}$ **Configuration stepping**: "Configuration $\mathcal{C}_1$ steps to $\mathcal{C}_2$"

STPC-LOCAL	$M\{\sigma_1;\kappa;\psi_1;e_1\}$	$\longrightarrow$	$M\{\sigma_2;\kappa;\psi_2;e_2\}$ when $\sigma_1;\psi_1;e_1 \xrightarrow{M} \sigma_2;\psi_2;e_2$
STPC-LET	$M\{\sigma;\kappa;\psi;\textbf{let}\ x = e_1\ \textbf{in}\ e_2\}$	$\longrightarrow$	$M\{\sigma;\kappa :: \langle M;\psi;x.e_2\rangle;\psi;e_1\}$
STPC-DELPAR	$\mathsf{p}(w_1 \cup w_2)\{\sigma;\kappa;\psi;\textbf{let}\ x \overset{\mathsf{p}(w')}{=} e_1\ \textbf{in}\ e_2\}$	$\longrightarrow$	$\mathsf{p}(w_2)\{\sigma;\kappa :: \langle \mathsf{p}(w_1 \cup w_2);\psi;x.e_2\rangle;\psi;e_1\}$ when $\psi(w') = w_2$
STPC-DELSEC	$m(w)\{\sigma;\kappa;\psi;\textbf{let}\ x \overset{\mathsf{s}(w')}{=} e_1\ \textbf{in}\ e_2\}$	$\longrightarrow$	$m(w)\{\sigma;\kappa :: \langle m(w);\psi;x.e_2\rangle;\psi;\textbf{secure}_{w'}(e_1)\}$
STPC-SECENTER	$m(w)\{\sigma;\kappa;\psi;\textbf{secure}_w(e)\}$	$\longrightarrow$	$\mathsf{s}(w)\{\sigma;\kappa;\psi;e\}$
STPC-POPSTK	$N\{\sigma;\kappa :: \langle M;\psi_1;x.e\rangle;\psi_2;v\}$	$\longrightarrow$	$M\{\sigma;\kappa;\psi_1\{x \mapsto_{m(w)} \psi_2(v)\};e\}$ when $M = m(\_)$ and $N = \_(w)$

$\boxed{\sigma_1;\psi_1;e_1 \xrightarrow{M} \sigma_2;\psi_2;e_2}$ **Local stepping**: "Under store σ_1 and environment ψ_1, expression e_1 steps at mode M to σ_2, ψ_2 and e_2"

STPL-CASE	$\sigma;\psi;\textbf{case}\,(v, x_1.e_1, x_2.e_2)$	$\xrightarrow{M}$	$\sigma;\psi\{x_i \mapsto_M v'\};e_i$	when $\psi(v) = \textbf{inj}_i\ v'$
STPL-FST	$\sigma;\psi;\textbf{fst}\,(v)$	$\xrightarrow{M}$	$\sigma;\psi;v_1$	when $\psi(v) = (v_1, v_2)$
STPL-SND	$\sigma;\psi;\textbf{snd}\,(v)$	$\xrightarrow{M}$	$\sigma;\psi;v_2$	when $\psi(v) = (v_1, v_2)$
STPL-APP	$\sigma;\psi_1;\tilde{e}$	$\xrightarrow{M}$	$\sigma;\psi_2;e$	when $\psi_1;\tilde{e} \xrightarrow{M}_{\mathsf{app}} \psi_2;e$
STPL-FIX	$\sigma;\psi;\textbf{fix}\,f.\lambda x.e$	$\xrightarrow{\mathsf{p}(w)}$	$\sigma;\psi';e$	when $\psi' = \psi\{f \mapsto_{\mathsf{p}(w)} \textbf{clos}\,(\psi;\textbf{fix}\,f.\lambda x.e)\}$
STPL-ARRAY	$\sigma;\psi;\textbf{array}\ v_1\ v_2$	$\xrightarrow{M}$	$\sigma\{\ell :_M w^k\};\psi;\ell$	when $\psi(v_1) = k,\ \psi(v_2) = w$ and $\mathsf{next}_M(\sigma) = \ell$
STPL-SELECT	$\sigma;\psi;\textbf{select}\ v_1\ v_2$	$\xrightarrow{M}$	$\sigma;\psi;w_i$	when $\psi(v_1) = \ell,\ \psi(v_2) = i,\ 0 \le i \le k$ and $\sigma(\ell) = w_0, .., w_k$
STPL-SEL-ERR	$\sigma;\psi;\textbf{select}\ v_1\ v_2$	$\xrightarrow{M}$	$\sigma;\psi;\mathsf{error}$	when $\psi(v_1) = \ell,\ \psi(v_2) = i,\ i < 0 \vee k < i$ and $\sigma(\ell) = w_0, .., w_k$
STPL-UPDATE	$\sigma;\psi;\textbf{update}\ v_1\ v_2\ v_3$	$\xrightarrow{M}$	$\sigma';\psi;()$	when $\psi(v_1) = \ell,\ \psi(v_2) = i,\ \psi(v_3) = w_i'$ and $\sigma' = \sigma\{\ell :_M w_0', .., w_k'\}$ and $0 \le i \le k,\ w_i = w_i'$ for $i \neq j$ and $\sigma(\ell) = w_0, .., w_k$
STPL-UPD-ERR	$\sigma;\psi;\textbf{update}\ v_1\ v_2\ v_3$	$\xrightarrow{M}$	$\sigma;\psi;\mathsf{error}$	when $\psi(v_1) = \ell,\ \psi(v_2) = i,\ i < 0 \vee k < i$ and $\sigma(\ell) = w_0, .., w_k$
STPL-MAKESH	$\sigma;\psi;\textbf{makesh}\ v$	$\xrightarrow{\mathsf{s}(w)}$	$\sigma;\psi;\textbf{sh}\ w\ v'$	when $\psi(v) = v'$
STPL-COMBSH	$\sigma;\psi;\textbf{combsh}\ v$	$\xrightarrow{\mathsf{s}(w)}$	$\sigma;\psi;v$	when $\psi(v) = (\textbf{sh}\ w\ v)$
STPL-WIRE	$\sigma;\psi;\textbf{wire}_w(v)$	$\xrightarrow{M}$	$\sigma;\psi;\{\!|\psi(v)|\!\}^{\mathsf{wires}}_{\psi(w)}$	where $\{\!|v|\!\}^{\mathsf{wires}}_{w_1 \cup w_2} = \{\!|v|\!\}^{\mathsf{wires}}_{w_1} +\!\!+ \{\!|v|\!\}^{\mathsf{wires}}_{w_2}$
STPL-WCOPY	$\sigma;\psi;\textbf{wcopy}_w\ v$	$\xrightarrow{M}$	$\sigma;\psi;v$	and $\{\!|v|\!\}^{\mathsf{wires}}_{\{p\}} = \{p : v\}$ and $\{\!|v|\!\}^{\mathsf{wires}}_{\cdot} = \cdot$
STPL-PARPROJ	$\sigma;\psi;v_1[v_2]$	$\xrightarrow{\mathsf{p}(\{p\})}$	$\sigma;\psi;v'$	when $\psi(v_2) = p$ and $\psi(v_1) = \{p : v'\} +\!\!+ w'$
STPL-SECPROJ	$\sigma;\psi;v_1[v_2]$	$\xrightarrow{\mathsf{s}(\{p\} \cup w)}$	$\sigma;\psi;v'$	when $\psi(v_2) = p$ and $\psi(v_1) = \{p : v'\} +\!\!+ w'$
STPL-WAPP1	$\sigma;\psi;\textbf{wapp}_w(v_1, v_2)$	$\xrightarrow{M}$	$\sigma;\psi;\cdot$	when $\psi(w) = \cdot$
STPL-WAPP2	$\sigma;\psi;\textbf{wapp}_w(v_1, v_2)$	$\xrightarrow{M}$	$\sigma;\psi;e$	when $\psi(w) = \{p\} \cup w'$
	where $e = (\textbf{let}\ z_1 \overset{\mathsf{p}(\{p\})}{=} v_1[p]\ \textbf{in let}\ z_2 \overset{\mathsf{p}(\{p\})}{=} v_2[p]\ \textbf{in let}\ z_3 \overset{\mathsf{p}(\{p\})}{=} z_2\ z_1\ \textbf{in let}\ z_4 = \textbf{wapp}_{w'}(v_1, v_2)\ \textbf{in}\ ((\textbf{wire}_{\{p\}}(z_3)) +\!\!+ z_4))$			
STPL-WAPS1	$\sigma;\psi;\textbf{waps}_w(v_1, v_2)$	$\xrightarrow{M}$	$\sigma;\psi;\cdot$	when $\psi(w) = \cdot$
STPL-WAPS2	$\sigma;\psi;\textbf{waps}_w(v_1, v_2)$	$\xrightarrow{M}$	$\sigma;\psi;e$	when $\psi(w) = \{p\} \cup w'$
				where $e = (\textbf{let}\ z_1 = v_1[p]\ \textbf{in let}\ z_2 = v_2\ z_1\ \textbf{in let}\ z_3 = \textbf{waps}_{w'}(v_1, v_2)\ \textbf{in}\ ((\textbf{wire}_{\{p\}}(z_2)) +\!\!+ z_3))$
STPL-WFOLD1	$\sigma;\psi;\textbf{wfold}_w(v_1, v_2, v_3)$	$\xrightarrow{M}$	$\sigma;\psi;v_2$	when $\psi(w) = \cdot$
STPL-WFOLD2	$\sigma;\psi;\textbf{wfold}_w(v_1, v_2, v_3)$	$\xrightarrow{M}$	$\sigma;\psi;e$	when $\psi(w) = \{p\} \cup w'$
				where $e = (\textbf{let}\ z_1 = v_1[p]\ \textbf{in let}\ z_2 = v_3\ v_2\ p\ z_1\ \textbf{in wfold}_{w'}(v_1, z_2, v_3))$

$\boxed{\psi_1;\tilde{e} \xrightarrow{M}_{\mathsf{app}} \psi_2;e}$ **Application stepping**: "Under environment ψ_1 application $\tilde{e}$ steps at mode M to ψ_2 and e"

STPA-LAMBDA	$\psi;\lambda x.e$	$\xrightarrow{M}_{\mathsf{app}}$	$\psi;\lambda x.e$	
STPA-CLOS	$\psi;x$	$\xrightarrow{M}_{\mathsf{app}}$	$\psi';\lambda x.e$	when $\psi(x) = \textbf{clos}\,(\psi';\lambda x.e)$
STPA-APPLY	$\psi_1;\tilde{e}\ v$	$\xrightarrow{M}_{\mathsf{app}}$	$\psi_2\{x \mapsto_M v'\};e$	when $\psi_1;\tilde{e} \xrightarrow{M}_{\mathsf{app}} \psi_2;\lambda x.e$ and $\psi_1(v) = v'$

Fig. 5. λ_{Wy}: operational semantics of single-threaded configurations

Multi-Threaded Semantics

Protocols
Agents

$\pi ::= \varepsilon \mid \pi_1 \cdot \pi_2 \mid A$

$A ::= \mathsf{single}(p)\{\sigma;\kappa;\psi;e\}$
$\mid \mathsf{secure}(w_1 / w_2)\{\sigma;\kappa;\psi;e\}$

$\pi_1 \longrightarrow \pi_2$

Protocol stepping: "Protocol π_1 steps to π_2"

StpP-Private	$p\,\{\sigma_1;\kappa_1;\psi_1;e_1\}$	$\longrightarrow$	$p\,\{\sigma_2;\kappa_2;\psi_2;e_2\}$	when $\mathsf{p}(\{p\})\{\sigma_1;\kappa_1;\psi_1;e_1\} \longrightarrow \mathsf{p}(\{p\})\{\sigma_2;\kappa_2;\psi_2;e_2\}$
StpP-Present	$p\left\{\sigma;\kappa;\psi;\textbf{let}\ x \overset{\mathsf{p}(w)}{=} e_1\ \textbf{in}\ e_2\right\}$	$\longrightarrow$	$p\,\{\sigma;\kappa;\psi;\textbf{let}\ x = e_1\ \textbf{in}\ e_2\}$	when $\{p\} \subseteq \psi(w)$
StpP-Absent	$p\left\{\sigma;\kappa;\psi;\textbf{let}\ x \overset{\mathsf{p}(w)}{=} e_1\ \textbf{in}\ e_2\right\}$	$\longrightarrow$	$p\,\{\sigma;\kappa;\psi;\textbf{let}\ x = \bigcirc\ \textbf{in}\ e_2\}$	when $\{p\} \not\subseteq \psi(w)$
StpP-Frame	$\pi_1 \cdot \pi_2$	$\longrightarrow$	$\pi_1' \cdot \pi_2$	when $\pi_1 \longrightarrow \pi_1'$
StpP-SecBegin	ε	$\longrightarrow$	$\mathsf{s}(_w)\,\{\cdot;\cdot;\cdot;e\}$	
StpP-SecEnter	$\mathsf{s}(^{w_1}_{w_2})\,\{\sigma;\cdot;\psi;e\} \cdot p\,\{\sigma';\kappa;\psi';\textbf{secure}_{w_2}(e)\}$	$\longrightarrow$	$\mathsf{s}(^{w_1\cup\{p\}}_{w_2})\,\{\sigma\circ\sigma';\cdot;\psi\circ\psi';e\} \cdot p\,\{\sigma';\kappa;\cdot;\textbf{wait}\}$	
StpP-SecStep	$\mathsf{s}(^{w}_{w})\,\{\sigma_1;\kappa_1;\psi_1;e_1\}$	$\longrightarrow$	$\mathsf{s}(^{w}_{w})\,\{\sigma_2;\kappa_2;\psi_2;e_2\}$	when $\mathsf{s}(w)\{\sigma_1;\kappa_1;\psi_1;e_1\} \longrightarrow \mathsf{s}(w)\{\sigma_2;\kappa_2;\psi_2;e_2\}$
StpP-SecLeave	$\mathsf{s}(^{w_1\cup\{p\}}_{w_2})\,\{\sigma';\cdot;\psi;v\} \cdot p\,\{\sigma;\kappa;\cdot;\textbf{wait}\}$	$\longrightarrow$	$\mathsf{s}(^{w_1}_{w_2})\,\{\sigma';\cdot;\psi;v\} \cdot p\,\{\sigma;\kappa;\psi';v\}$	when $\mathsf{slice}_p(\psi) \rightsquigarrow \psi'$
StpP-SecEnd	$\mathsf{s}(_{w_2})\,\{\sigma;\cdot;\psi;v\}$	$\longrightarrow$	ε	

Fig. 6. λ_{Wy}: operational semantics of multi-threaded target protocols

store
stack
environment
program counter

Wysteria Meta Theory

- **Theorem** (Type soundness):
  - ***Well-typed programs make progress and preserve types***

- **Theorem** (Synchrony):
  - ***Single- & Multi-threaded operational views agree***

Type Soundness

Theorem [Progress]:

Suppose $\Gamma \vdash C_1 : \tau$

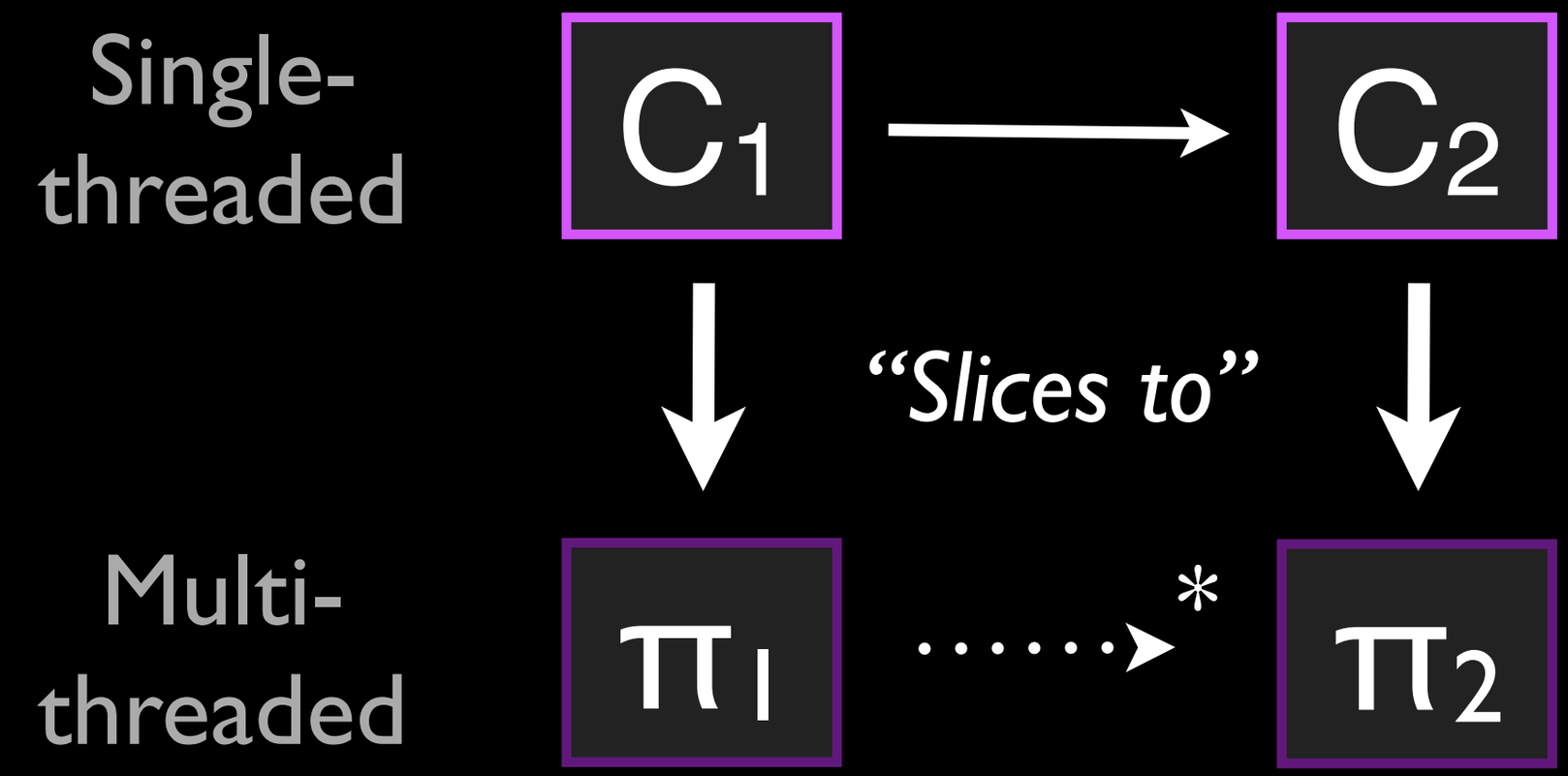
then exists C_2 such that $C_1 \rightarrow C_2$

Theorem [Preservation]:

Suppose $\Gamma \vdash C_1 : \tau$ and that $C_1 \rightarrow C_2$

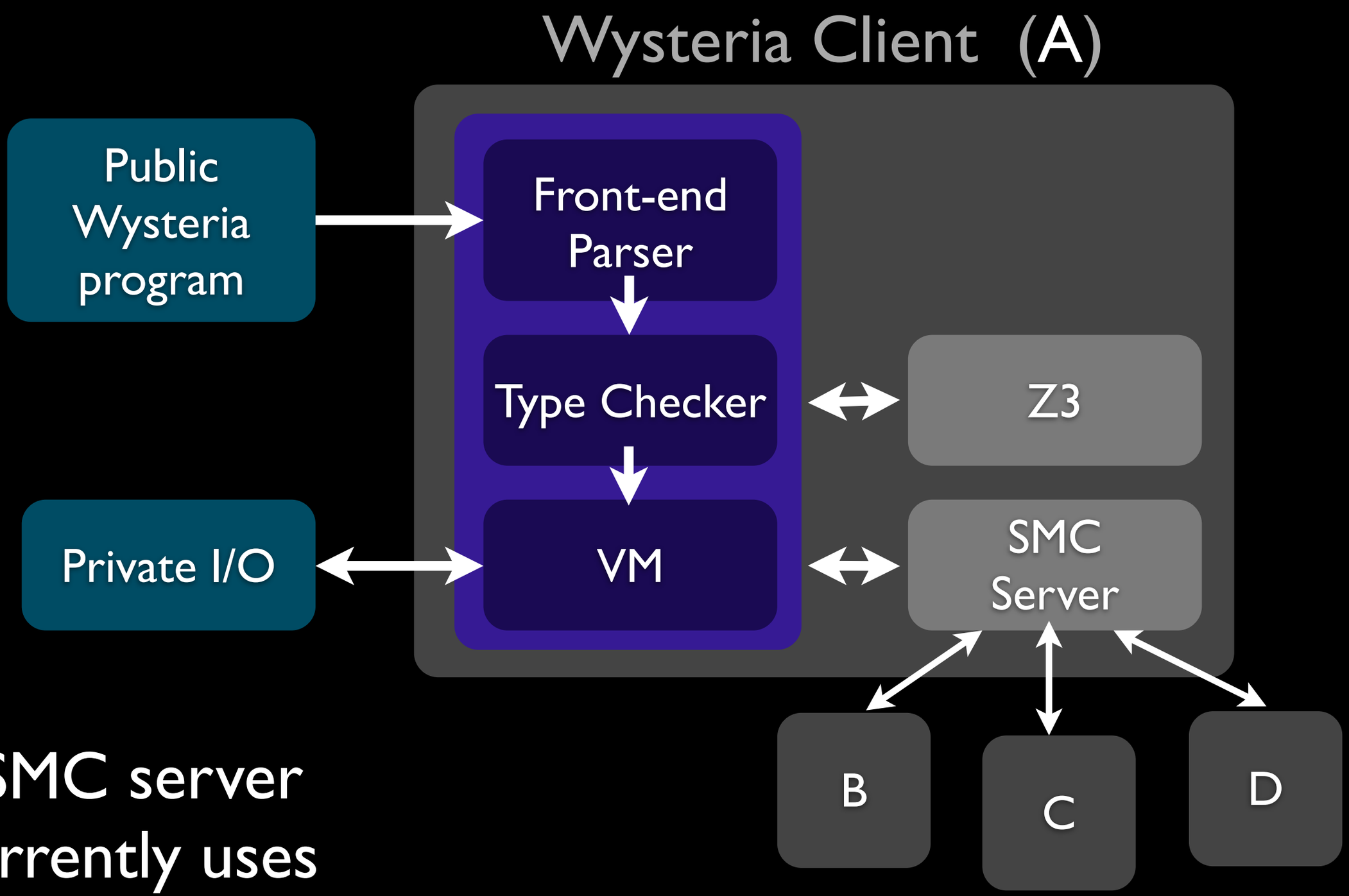
then $\Gamma \vdash C_2 : \tau$

Operational Synchrony



Wysteria System: Preliminary Evaluation

Prototype Implementation

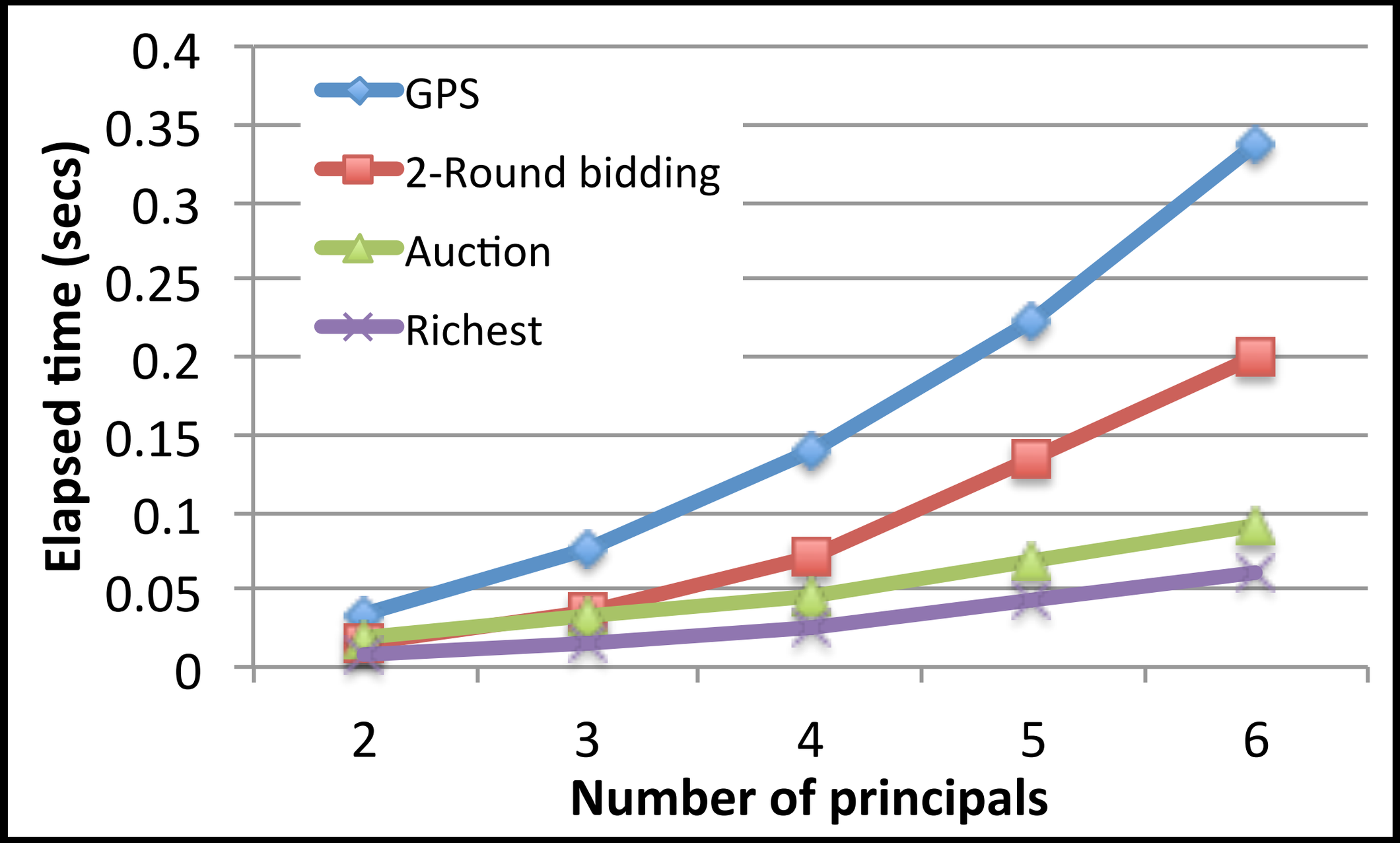


*SMC server currently uses GMW

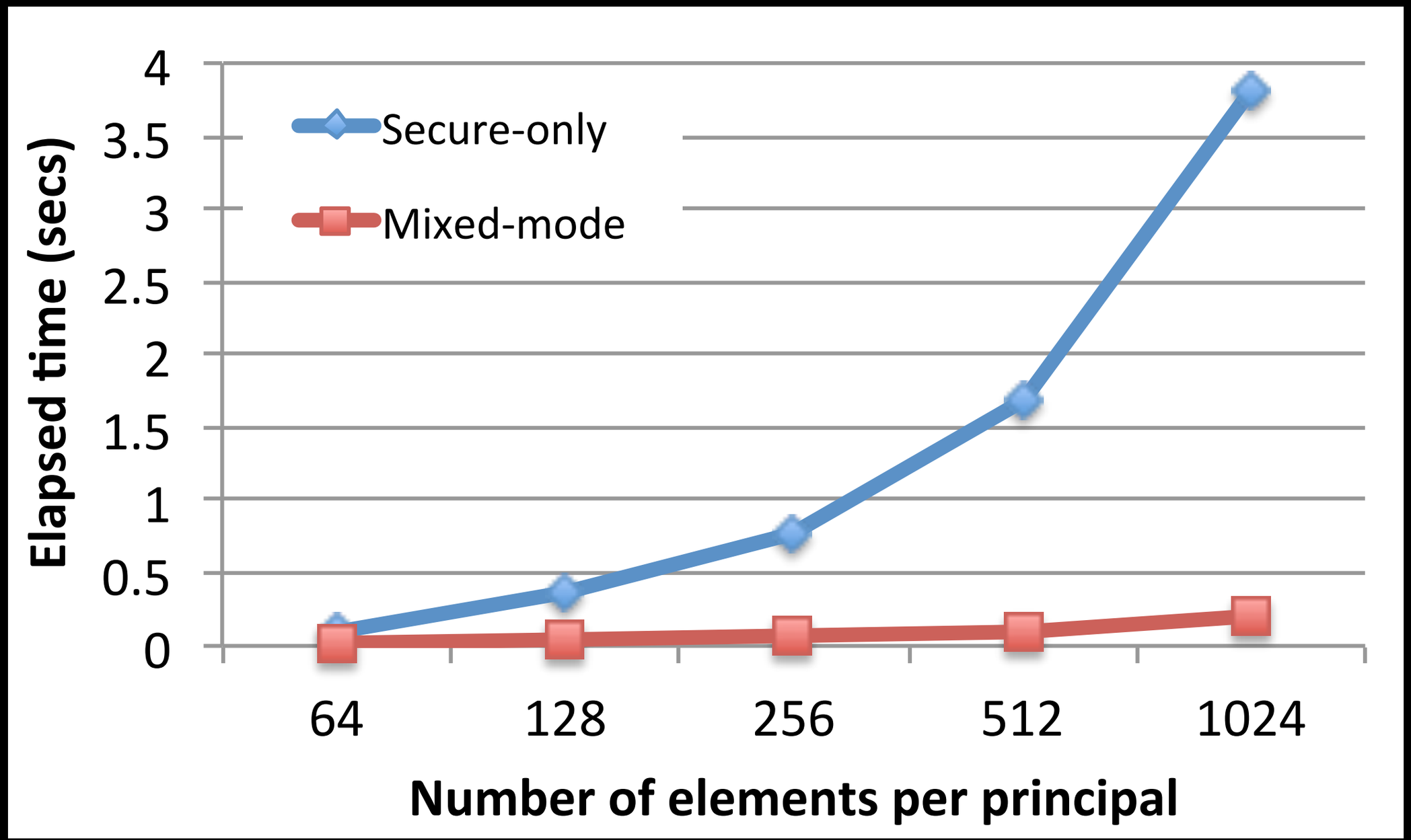
Implemented Examples

Application	*n* - ***party***?	***mixed***?
Millionaire's	yes	no
2nd-price auction	yes	no
2-round bidding	yes	yes
GPS	yes	no
Median	2-party	yes
PSI	2-party	yes
Card dealing	yes	yes

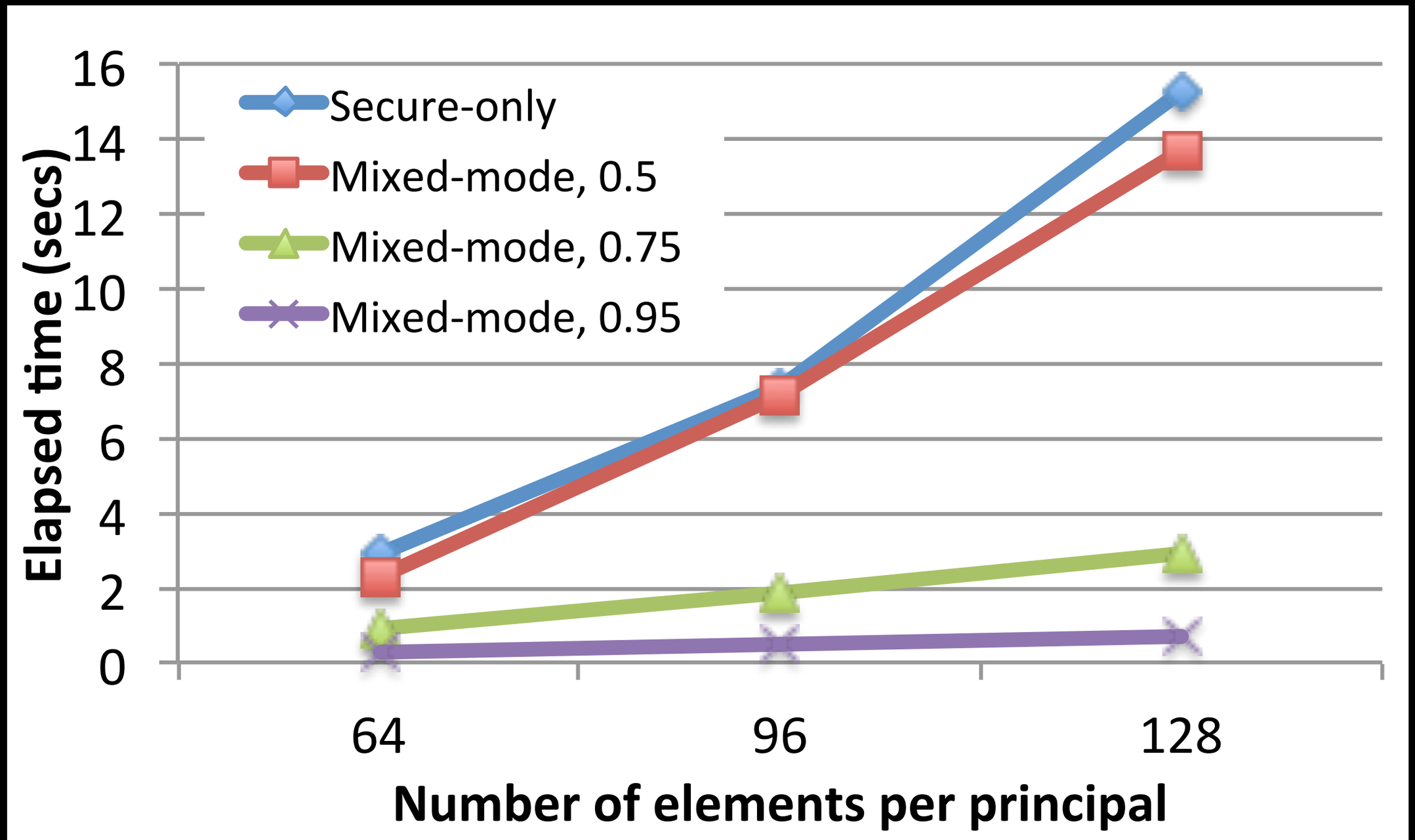
Multi-party Examples



Secure-only vs Mixed-mode (Two-party Median)



Secure-only vs Mixed-mode (Private set intersection)



Summary and Future Directions

Summary

- Secure Multiparty Computation is a useful abstraction
  - SMC simulates interaction with a trusted third party
- Wysteria gives compositional PL abstractions for SMC
- Protocols are (higher-order) functional programs
- Protocols can be **generic** in the involved principals
- Protocols can **mix modes**, capturing Reactive SMCs

Future Directions

- More Applications:
  - Card game library: Poker, Bridge, etc.
  - RTS game engine (?)
- Front-end enhancements:
  - Integration into OCaml (as an EDSL)
- Back-end enhancements:
  - Malicious attackers (now: Semi-honest)
  - Integration with Bitcoin:
    - Protocols with monetary transactions
    - Fairness, and disincentive for early quit