Advanced Process Calculi

Introduction to Psi-Calculi Workbench

Copenhagen, August 2013
Ramūnas Gutkovas

Psi-Calculi Workbench (Pwb)

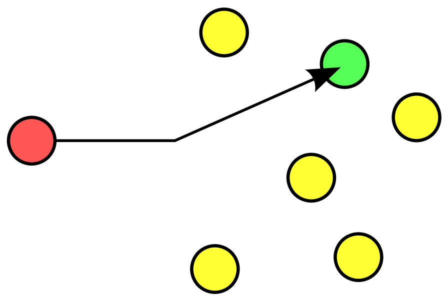
homepage: http://goo.gl/**ZJPu9**

- Tool for modeling concurrency
- Parametric:
    - Data, Logics, Logical Assertions
- Based on psi-calculi framework
- Free software

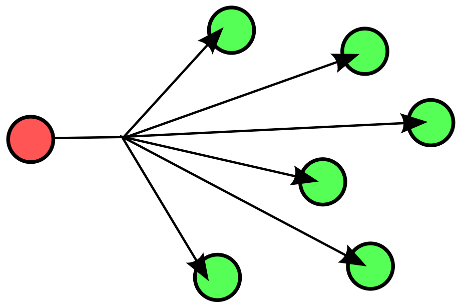
Features

Communication Primitives

Unicast



Wireless Broadcast



Parametric On

Data Structures

e.g., Names, Bits, Vectors, ADTs, Trees, ...

Logics

e.g., EUF, FOL, Equational Theory, ...

Logical Assertions

e.g., Knows a secret, Connectivity, Constraints...

Functionality

Symbolic Execution

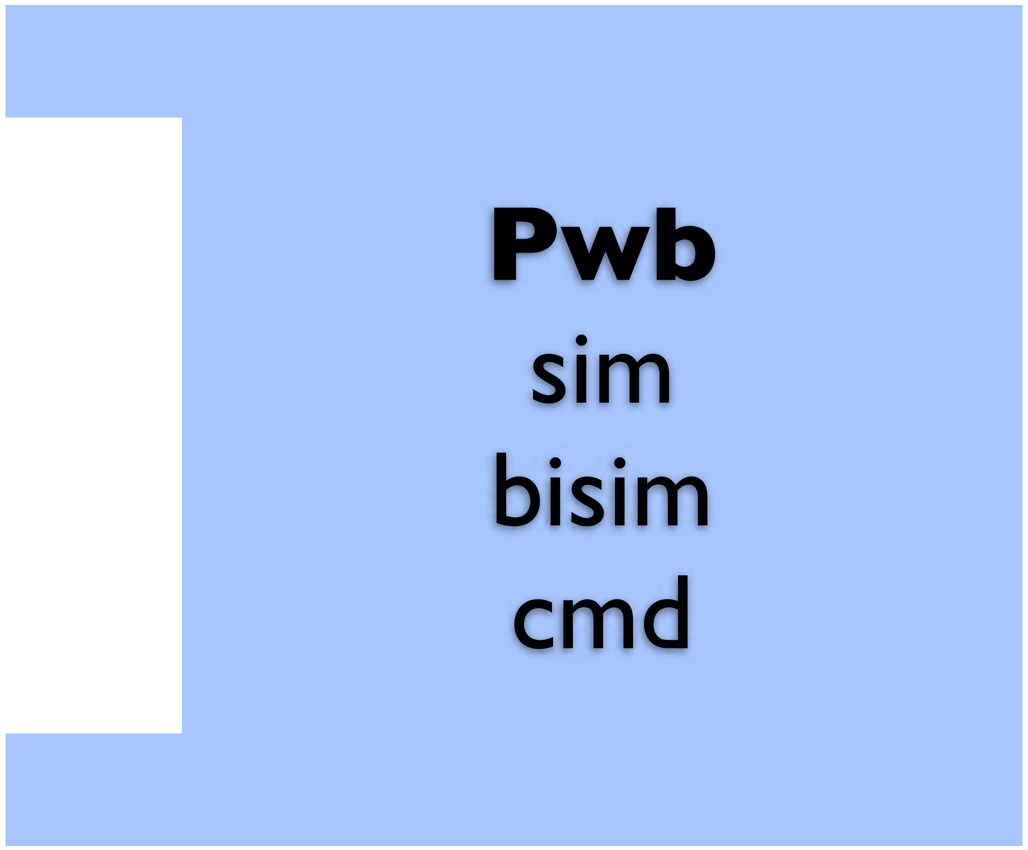
$$\Psi \triangleright P \xrightarrow[C]{\alpha} P'$$

Constraints

Symbolic Behavioral Equivalence Checking

$$P \sim Q$$

Tool Factory



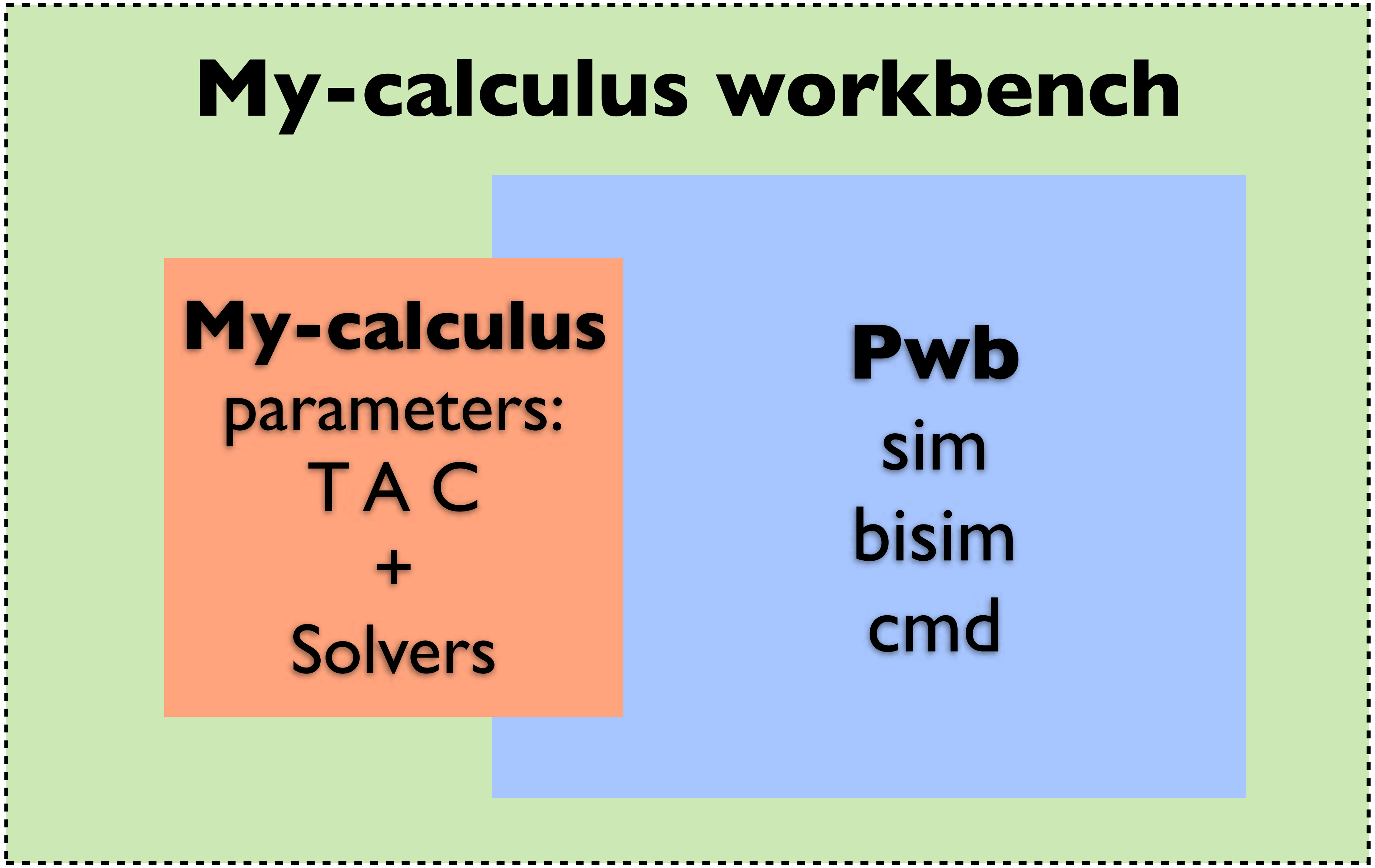
Pwb

sim

bisim

cmd

Tool Factory



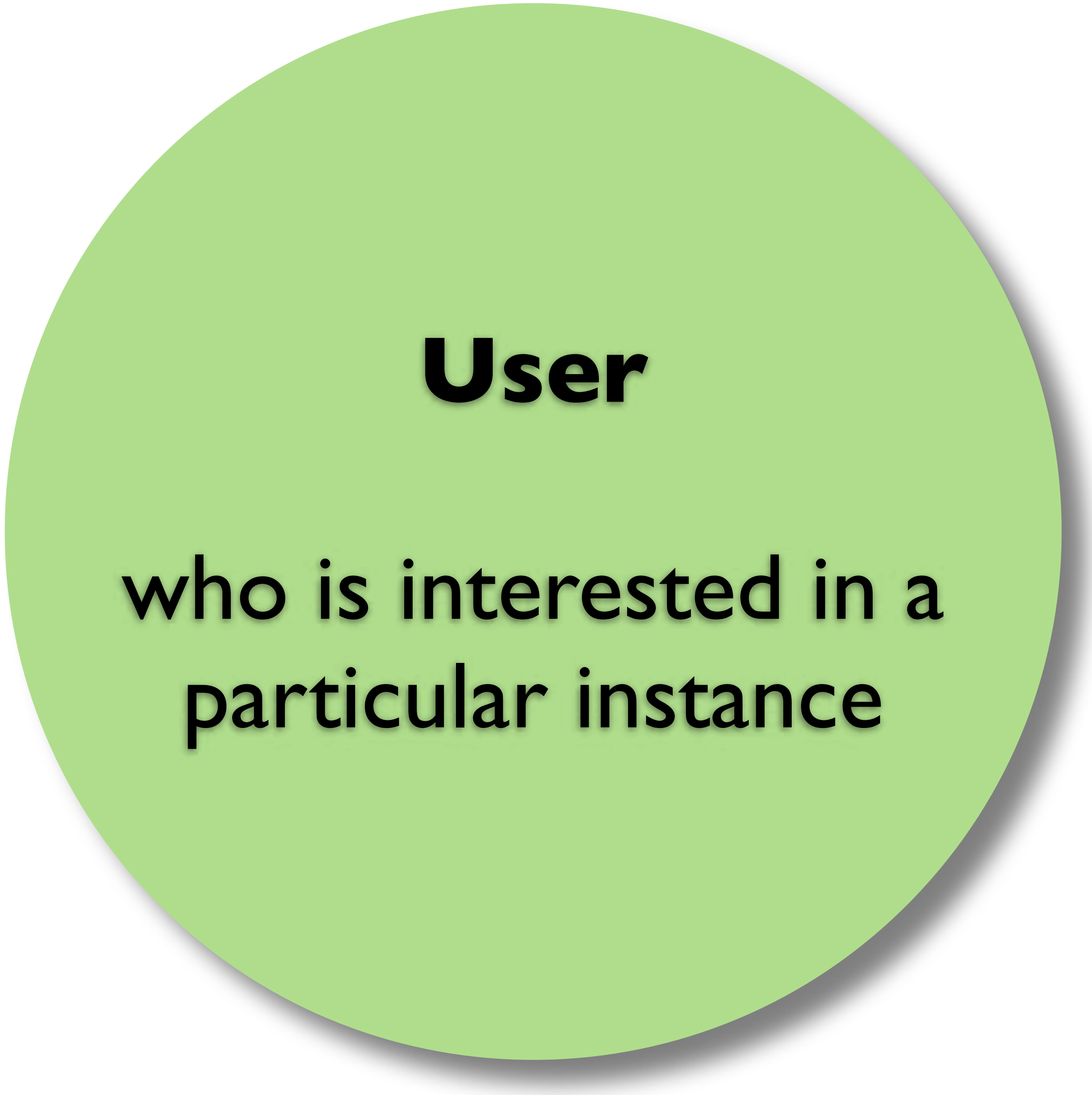
My-calculus workbench

My-calculus
parameters:
T A C
+
Solvers

Pwb
sim
bisim
cmd

Two Users

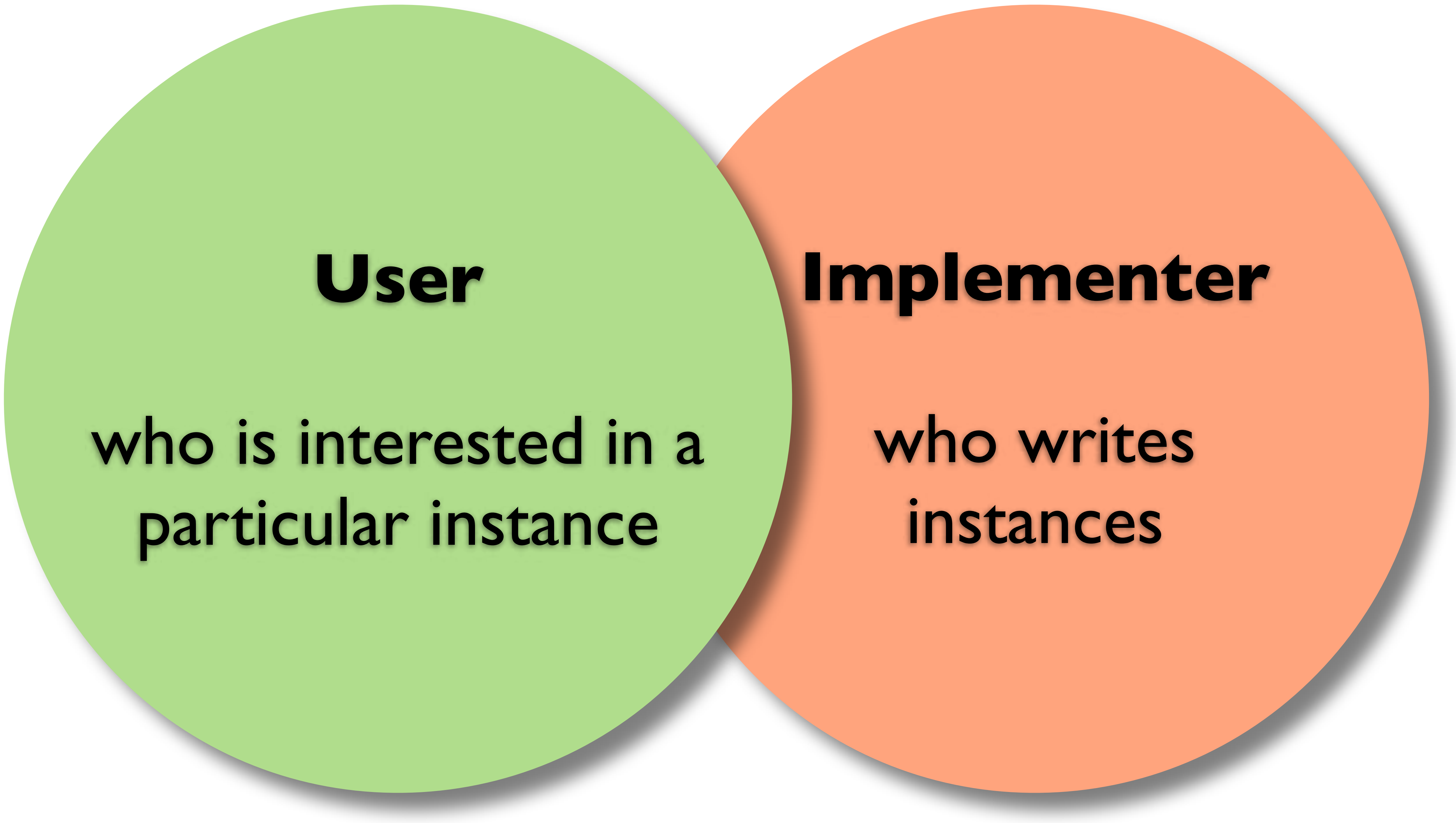
Two Users



User

who is interested in a particular instance

Two Users



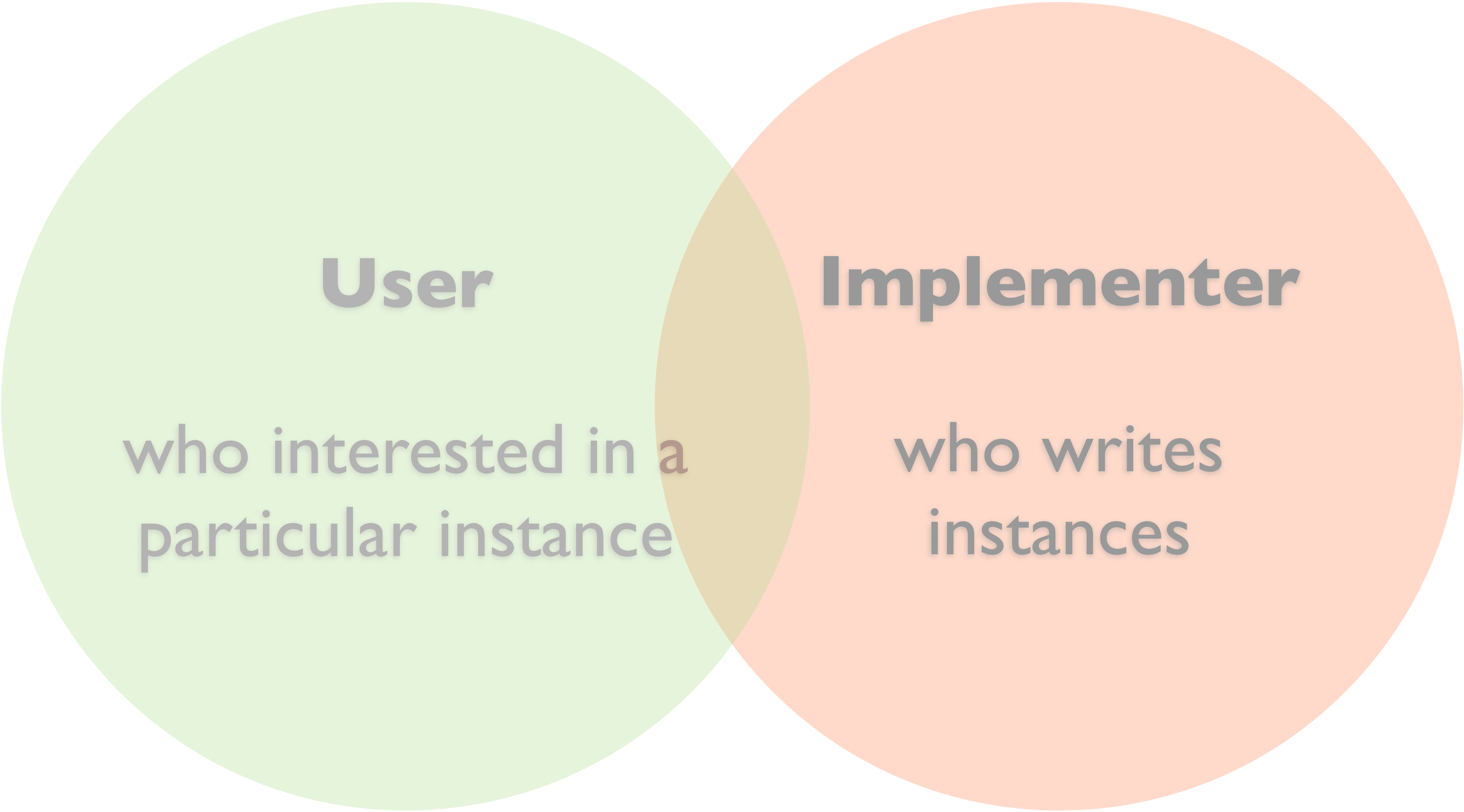
User

who is interested in a particular instance

Implementer

who writes instances

Two Users



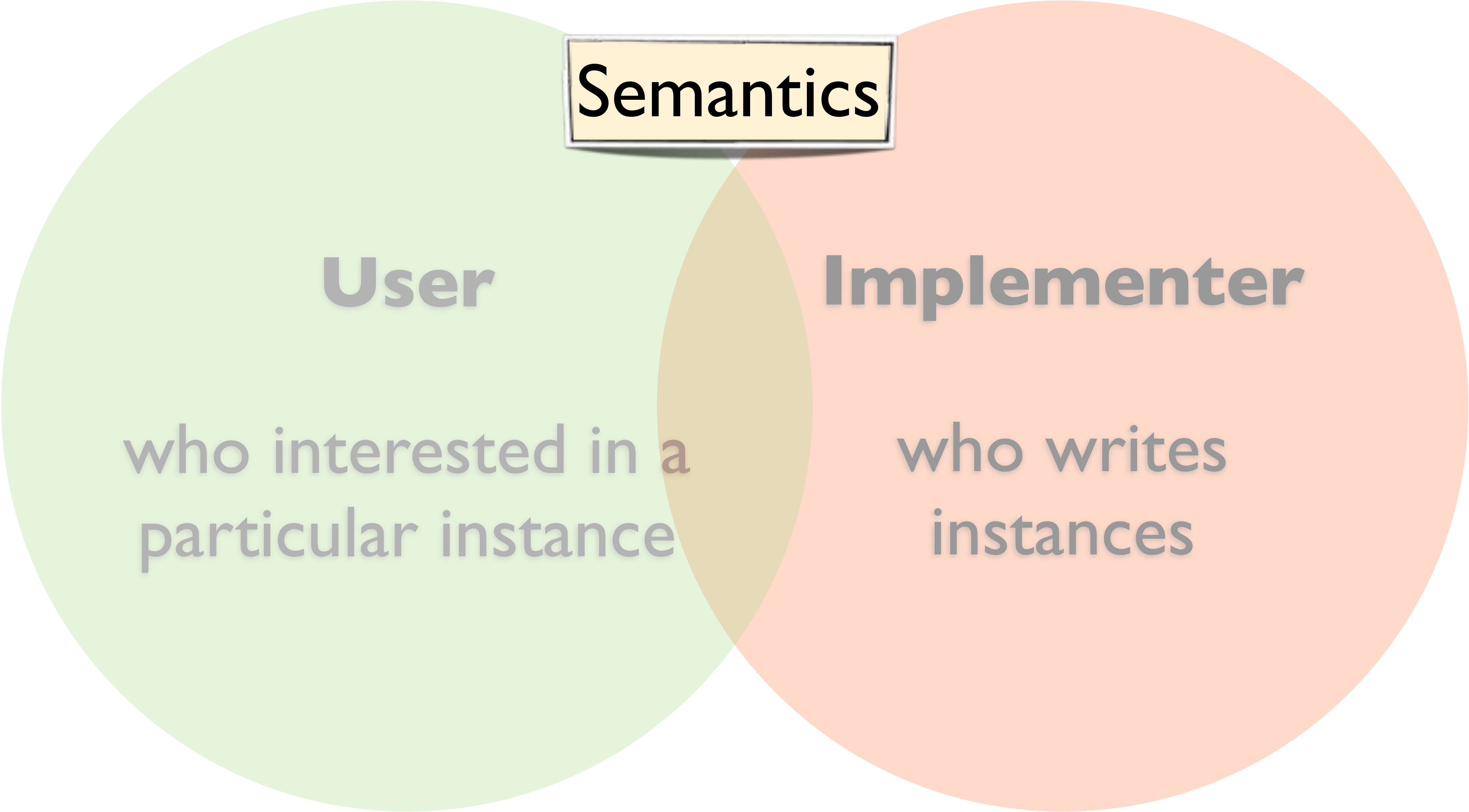
User

who interested in a particular instance

Implementer

who writes instances

Two Users



Semantics

User

who interested in a particular instance

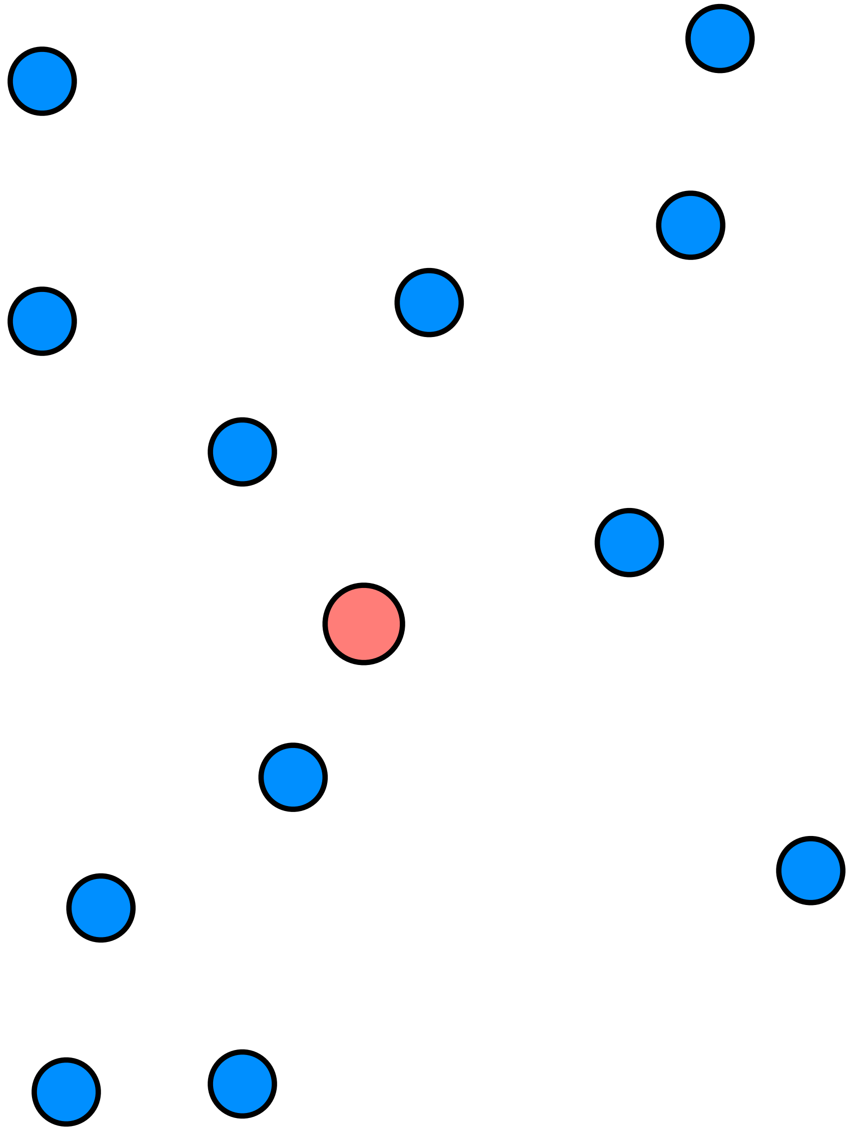
Implementer

who writes instances

Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

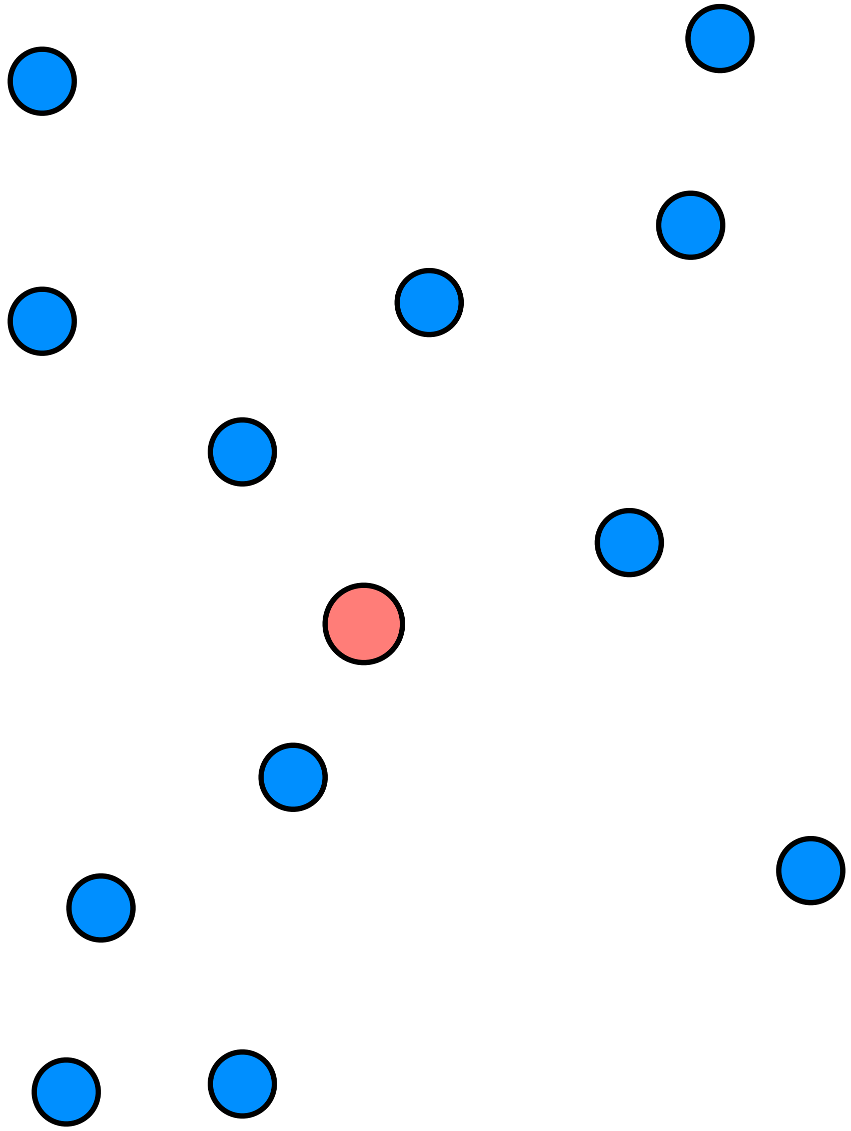
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

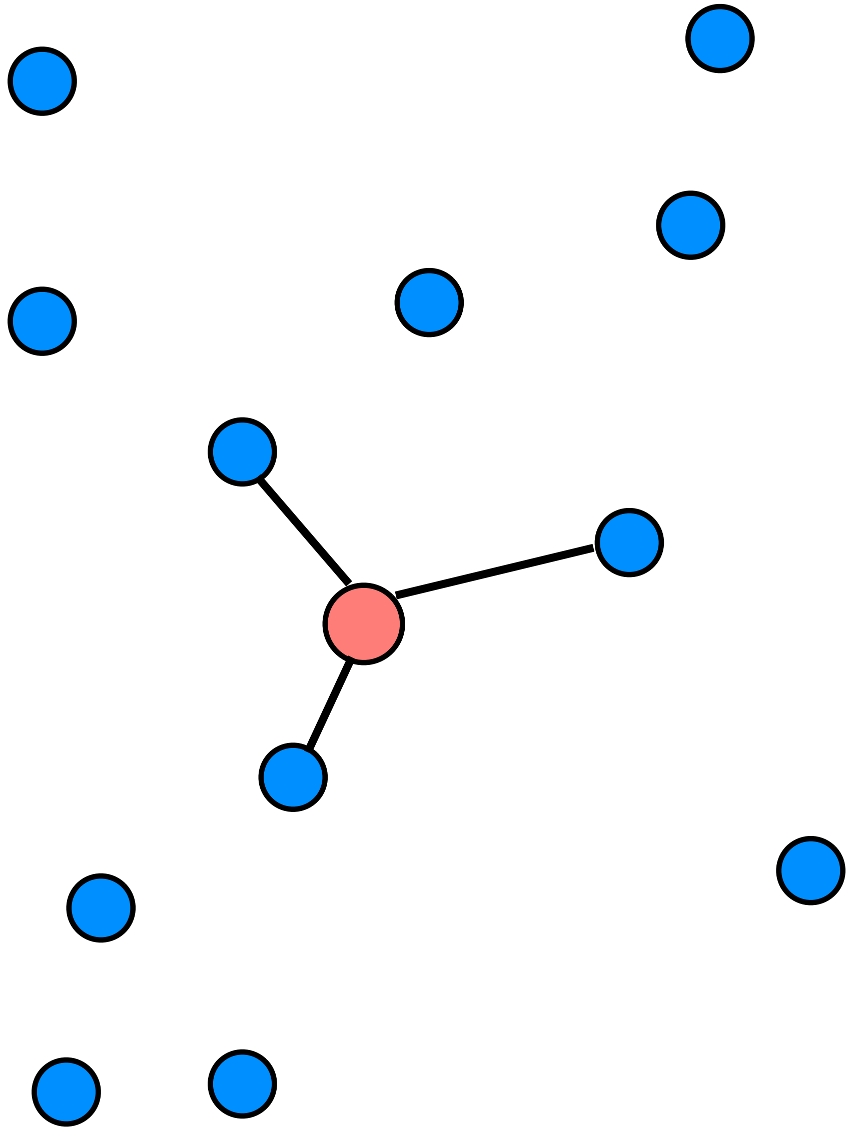
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

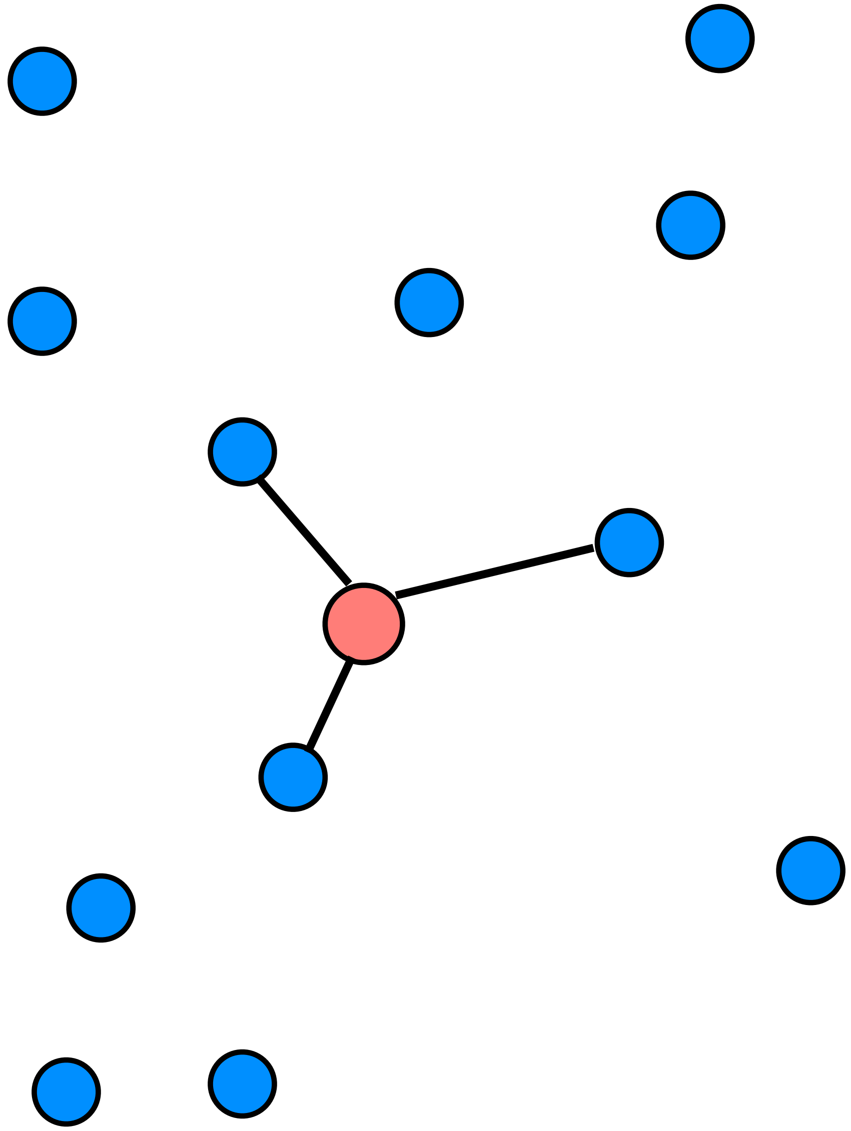
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

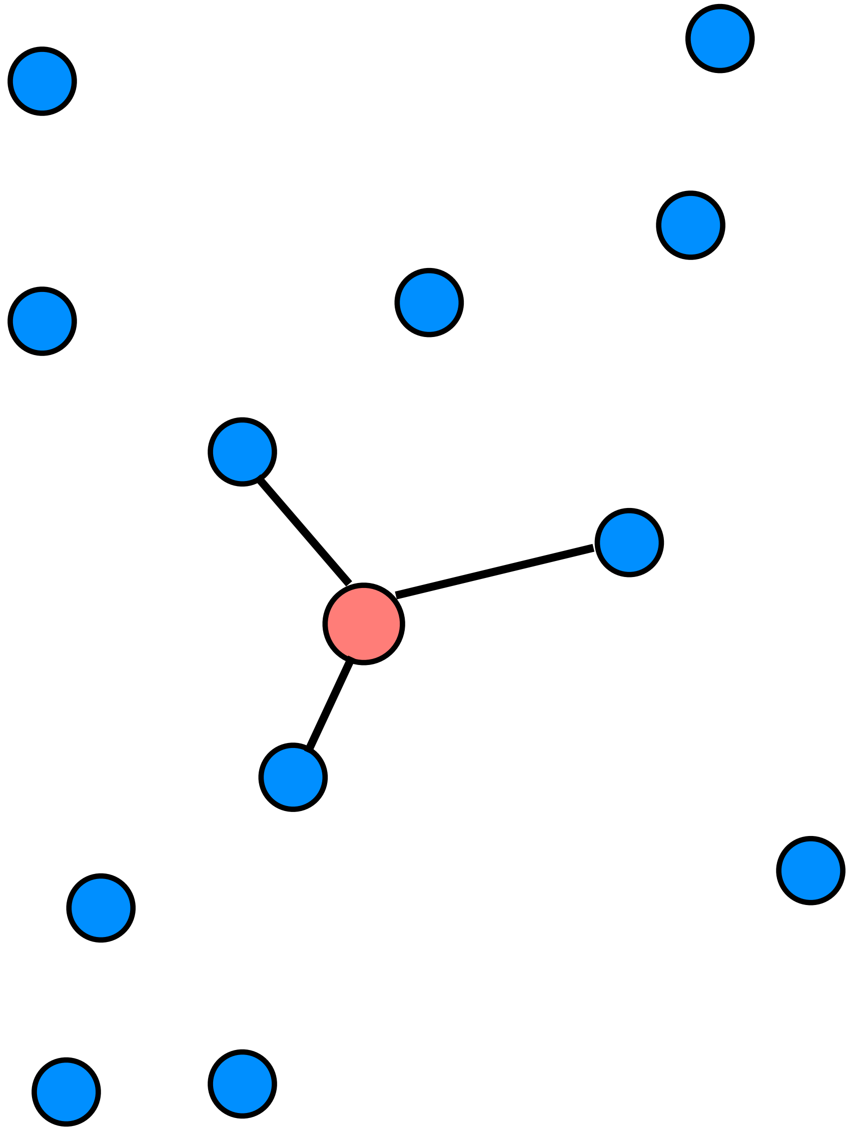
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

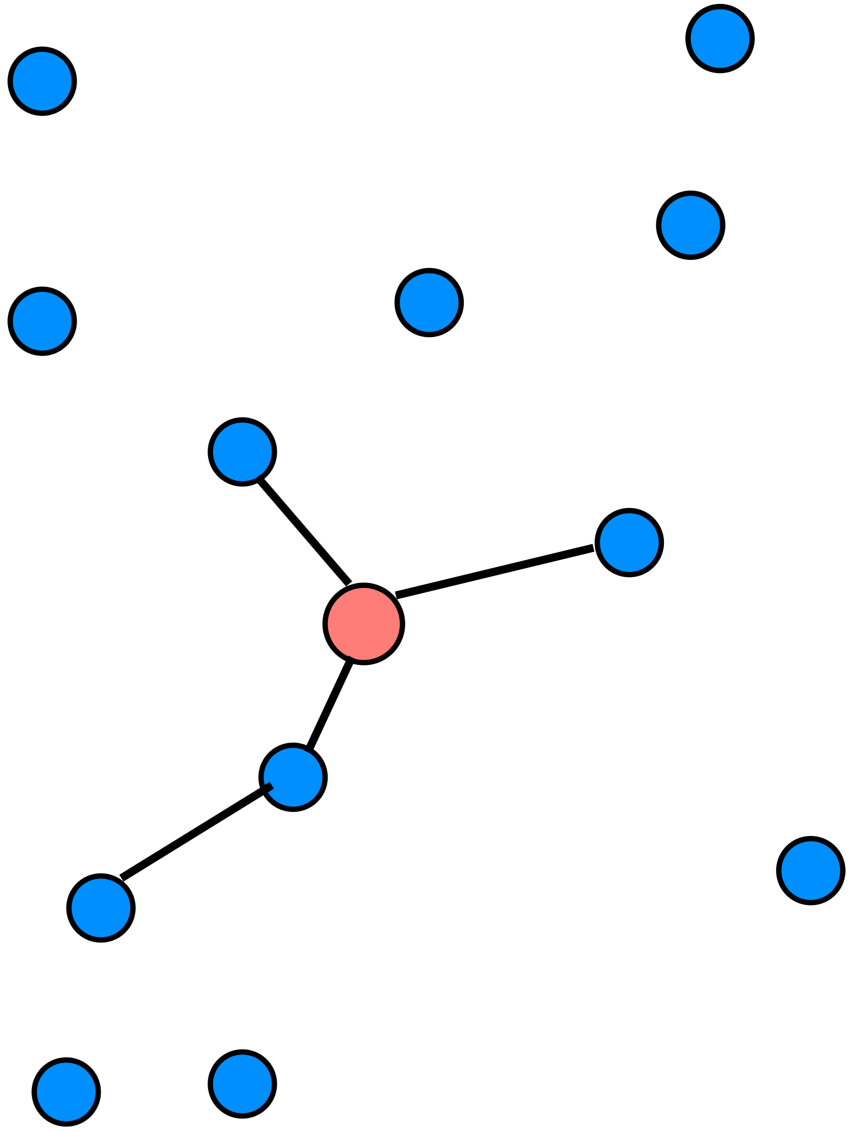
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

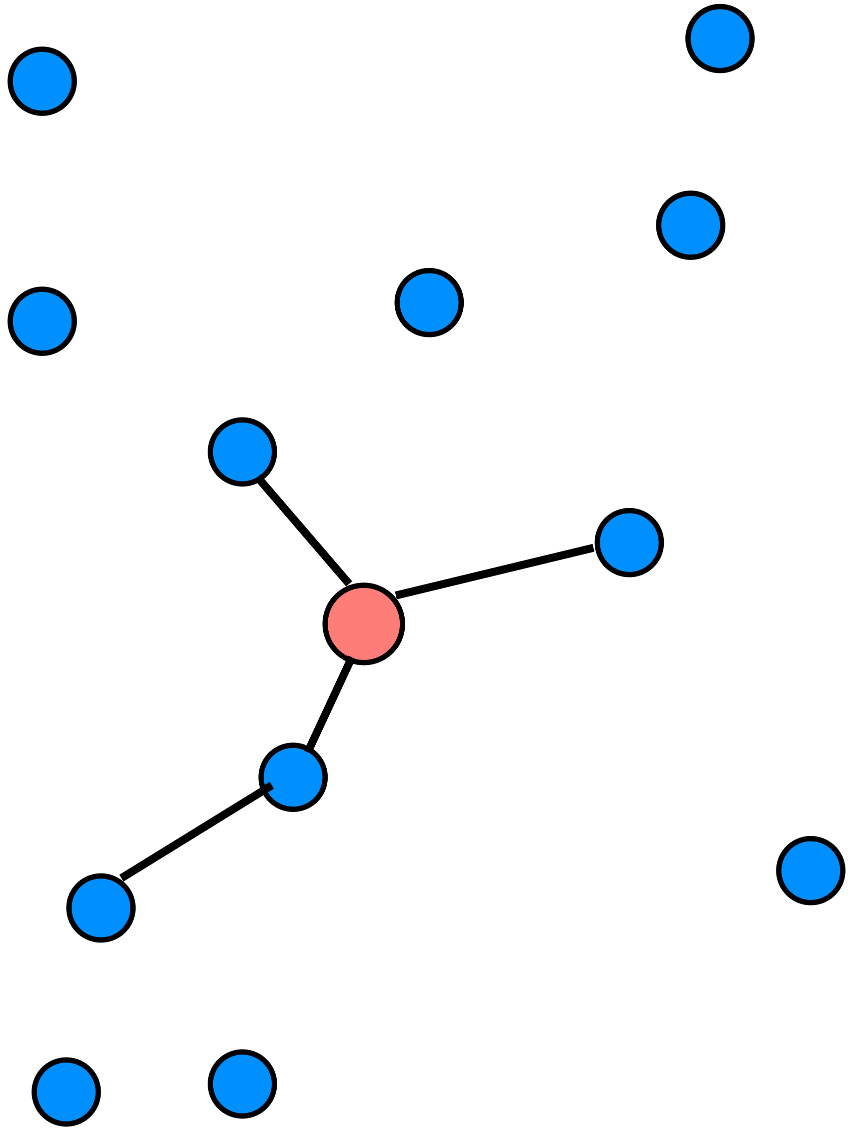
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

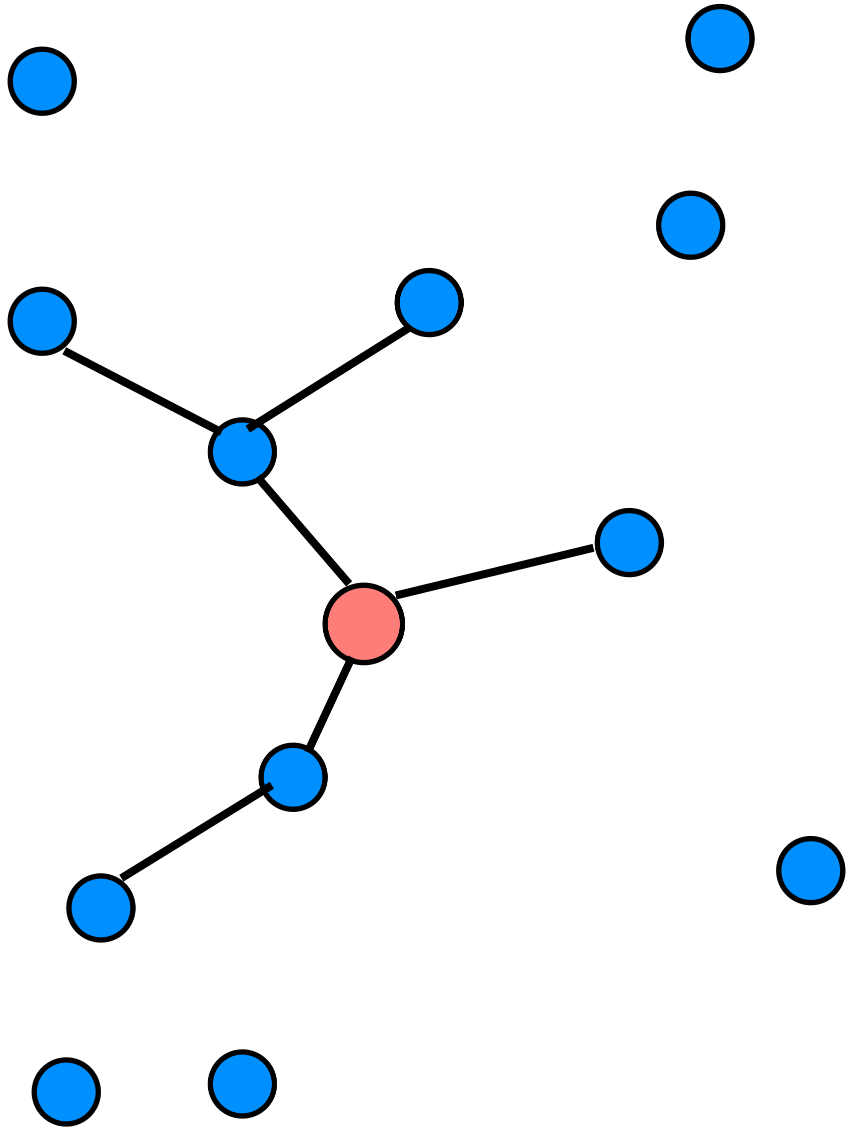
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

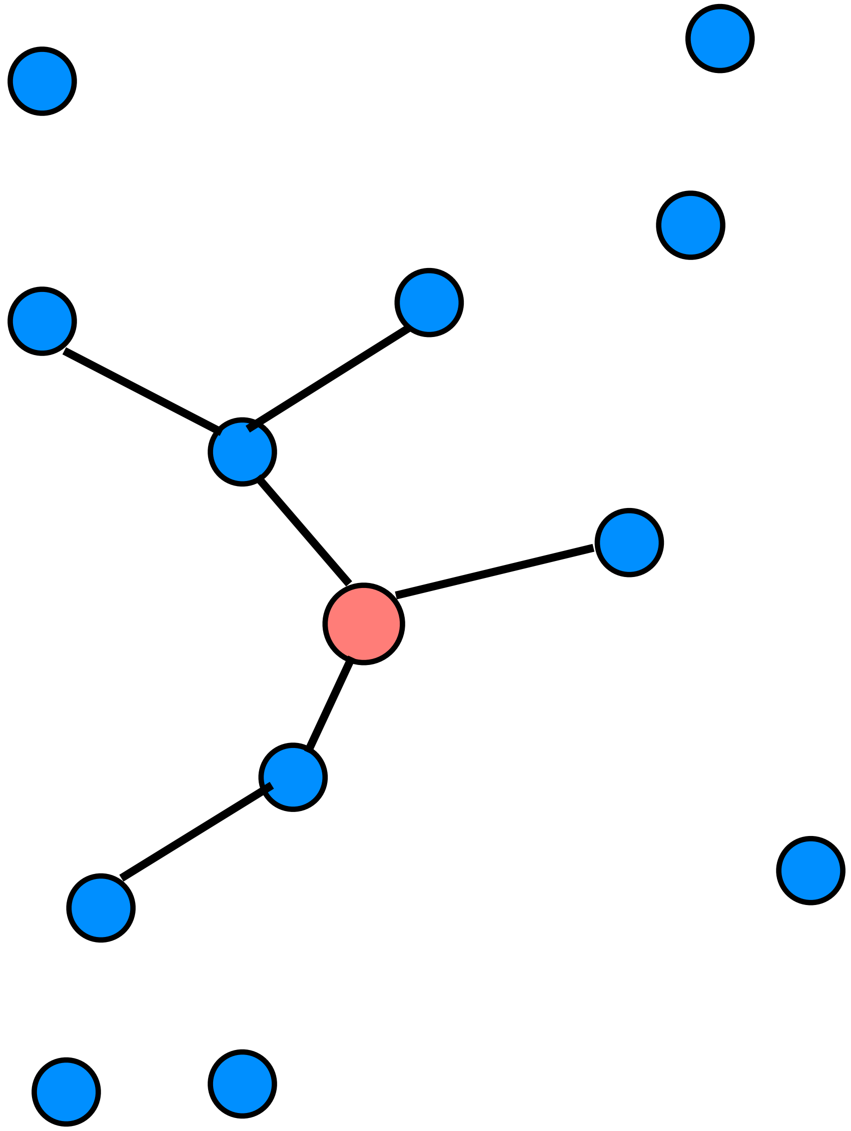
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

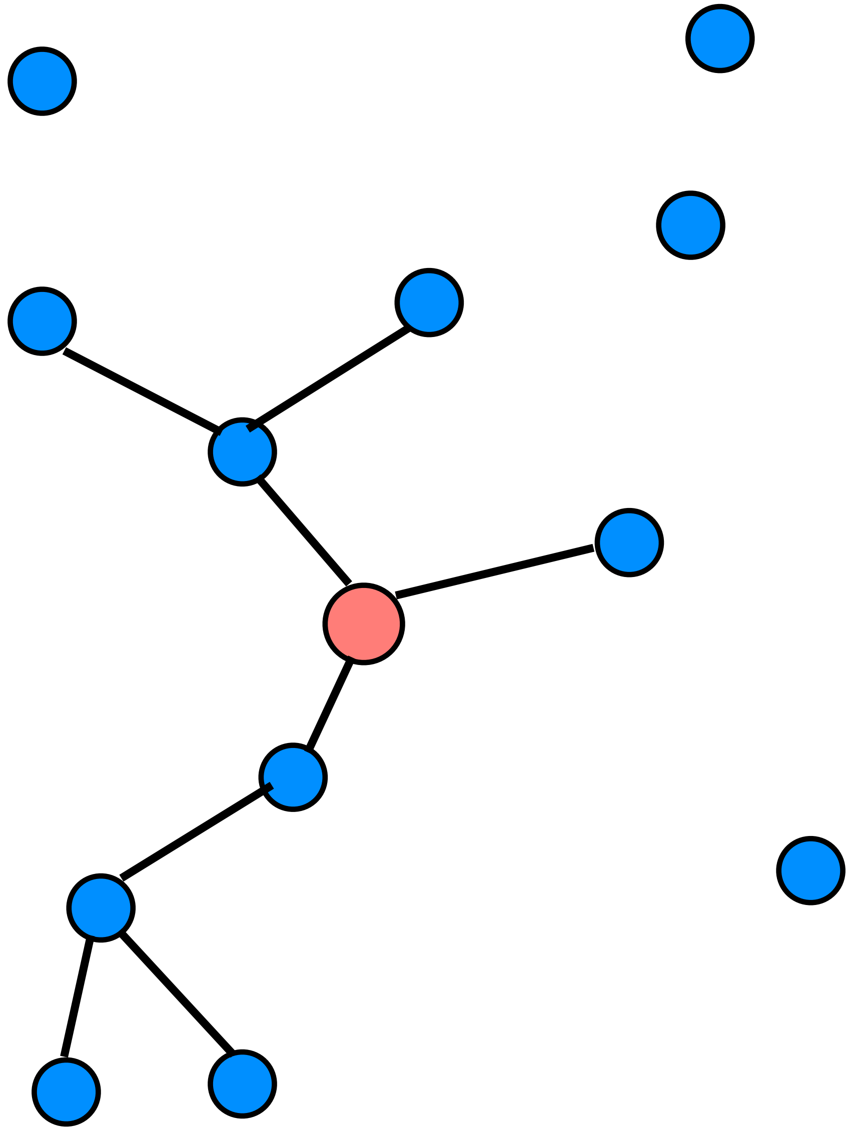
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

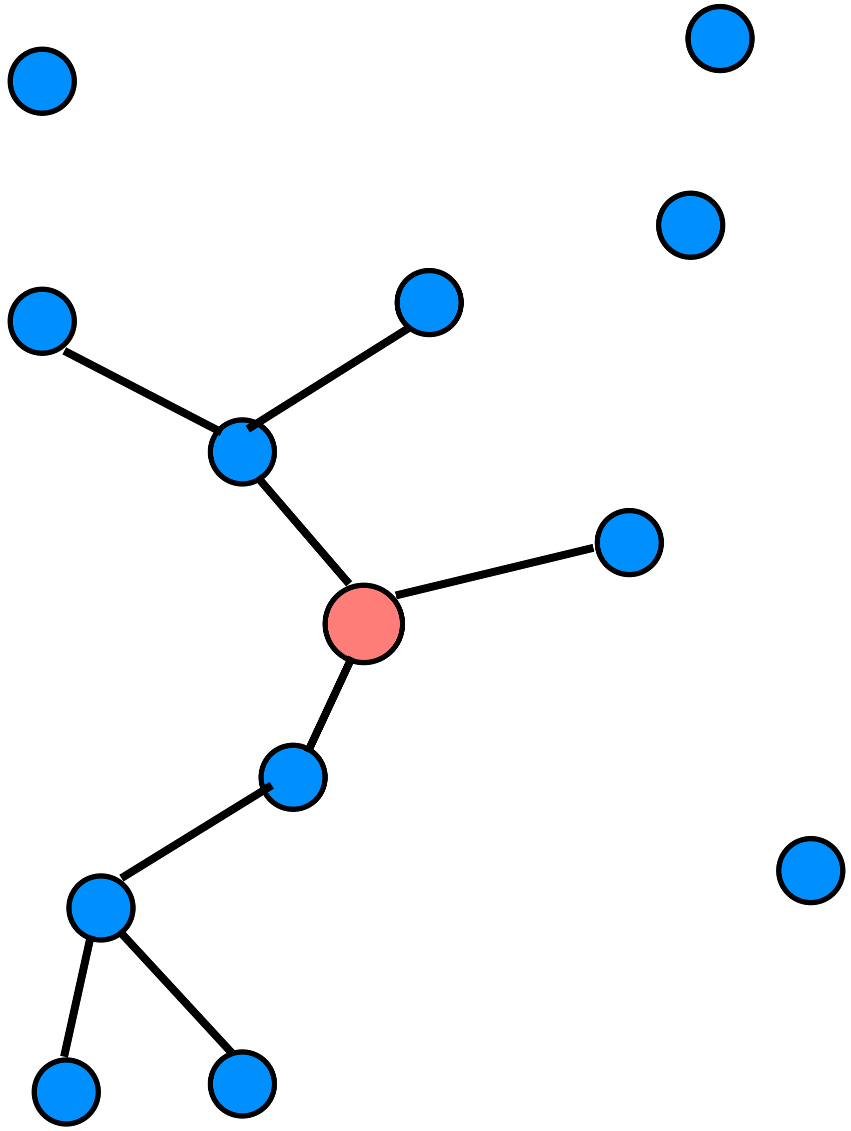
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

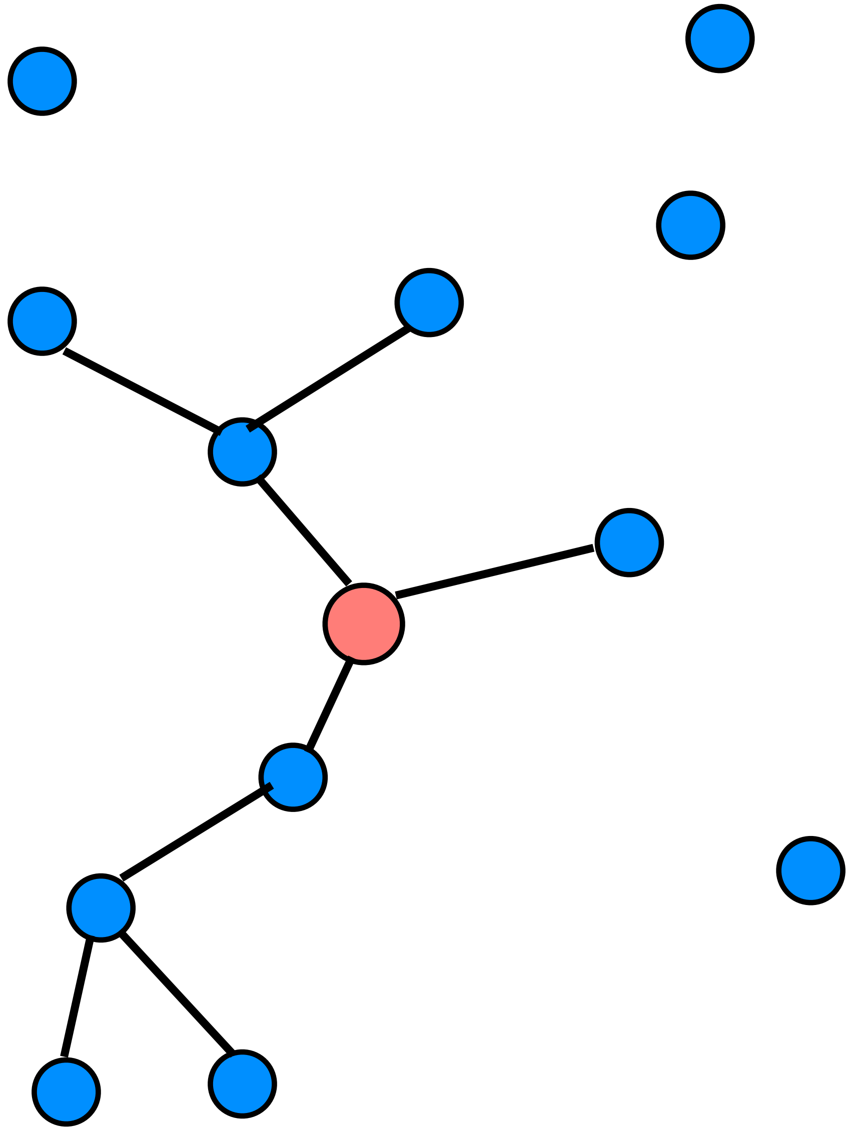
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

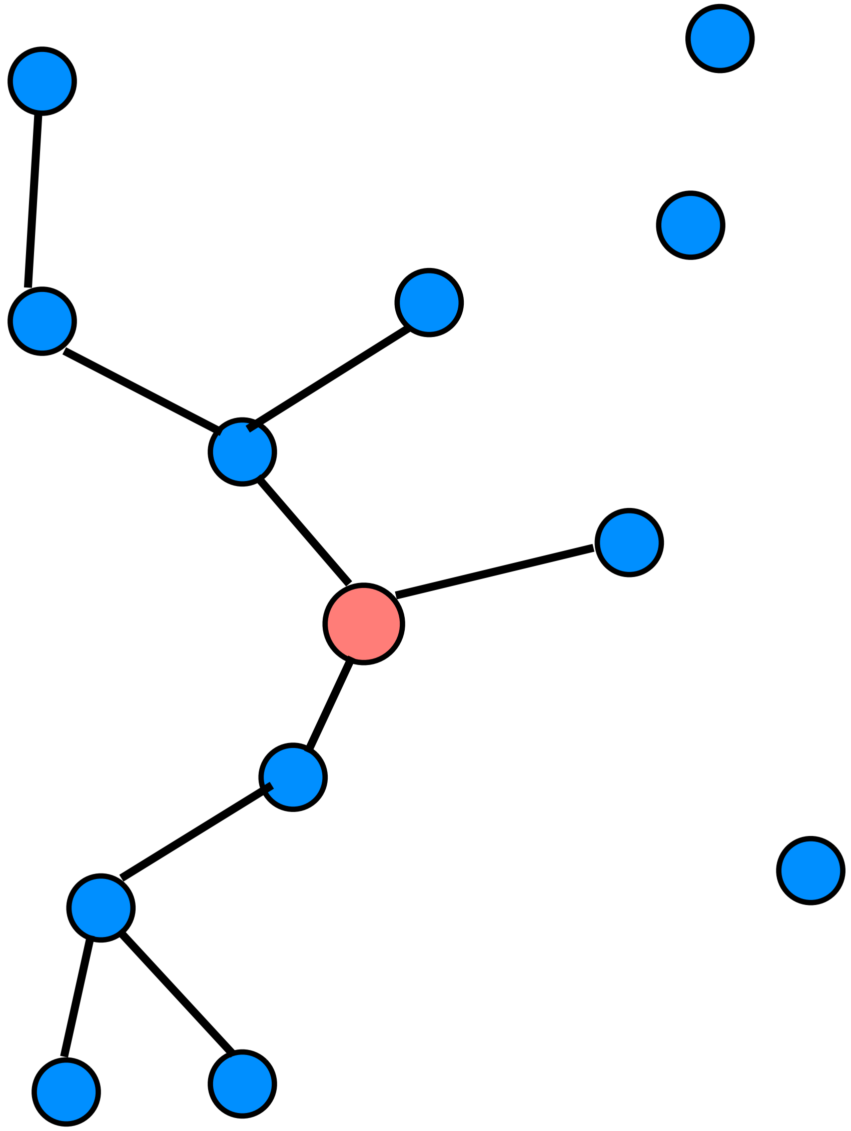
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

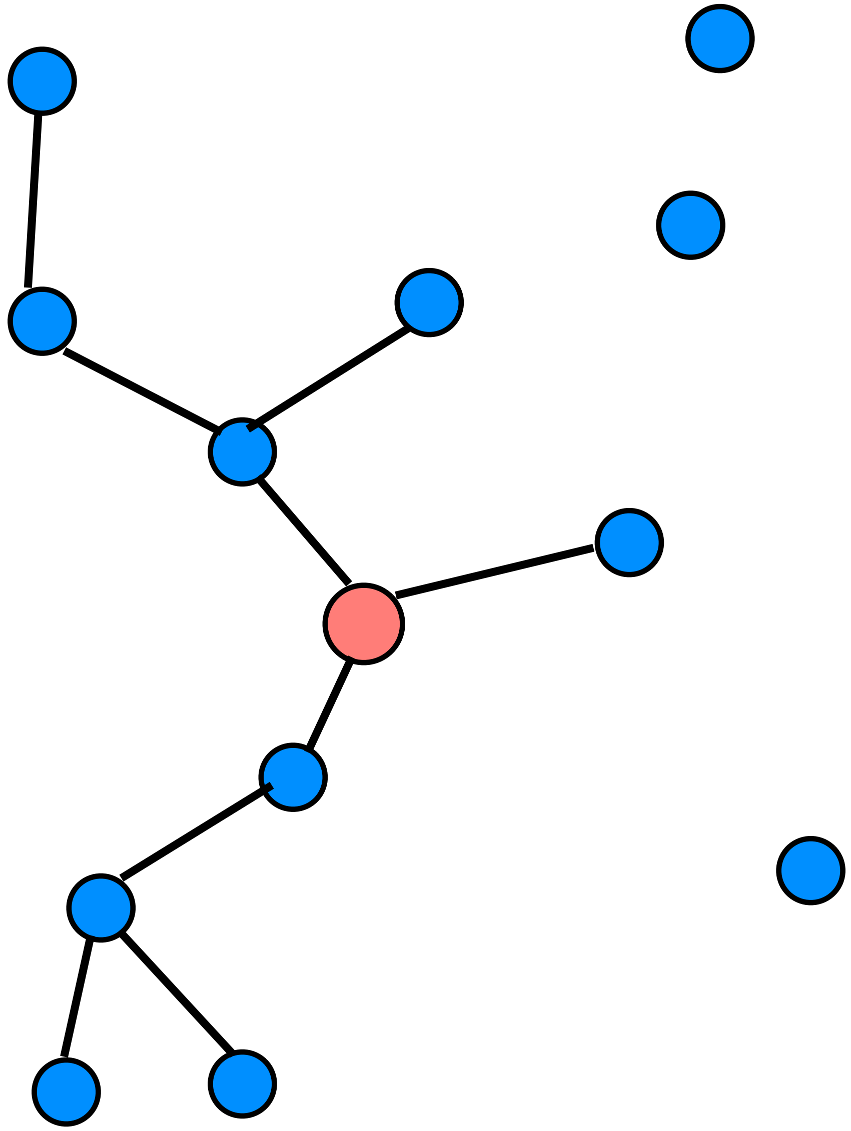
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

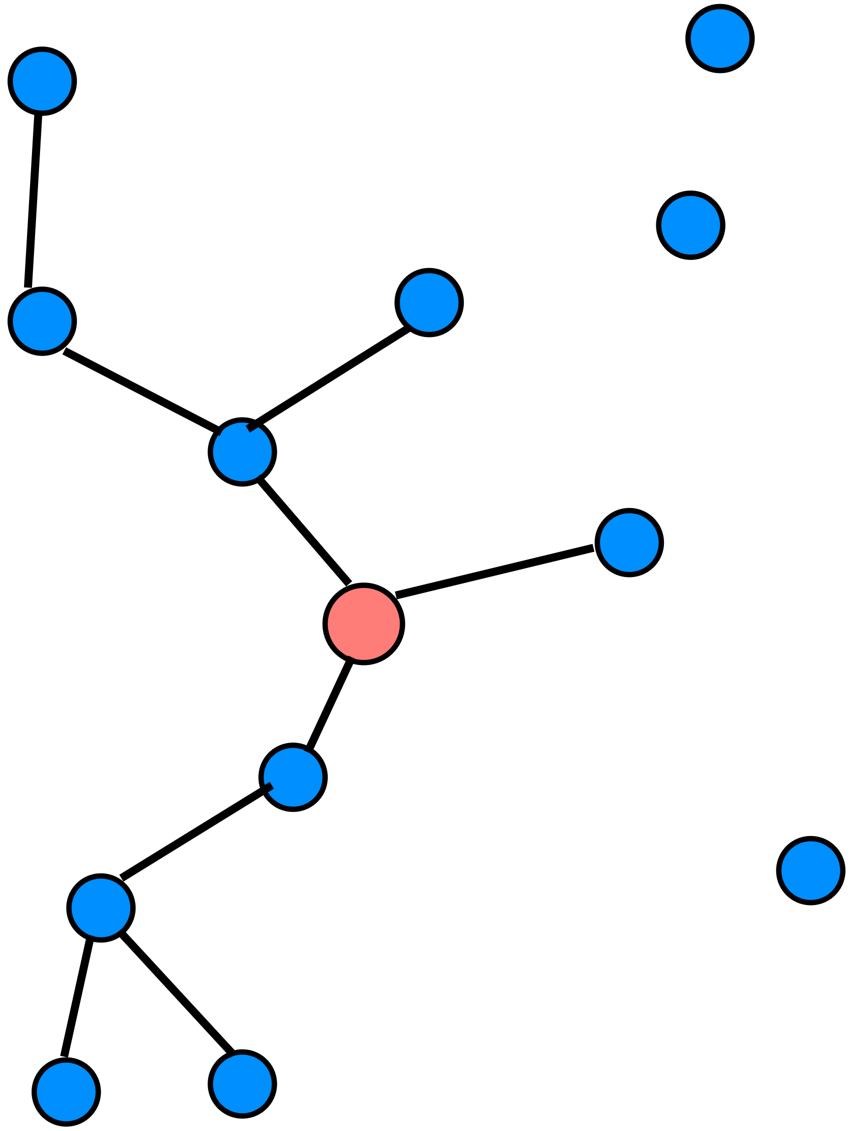
- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

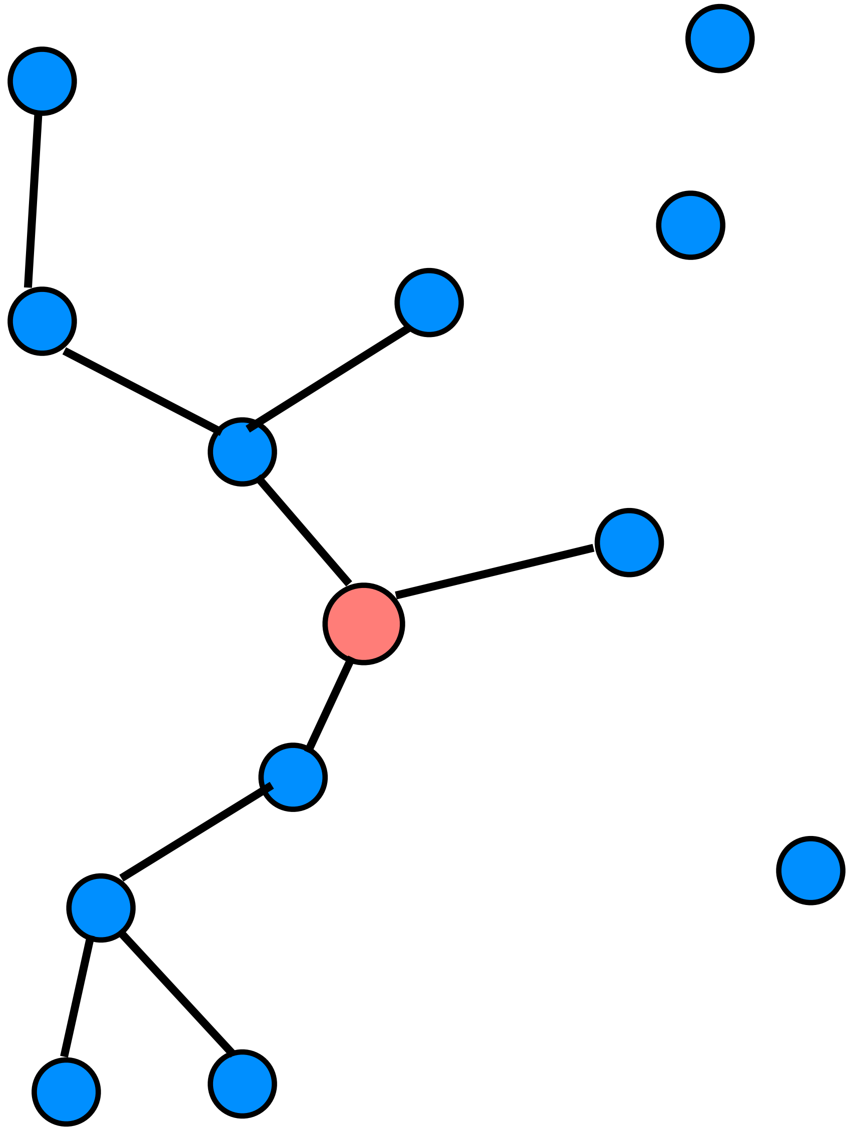
- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:
  1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
  2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

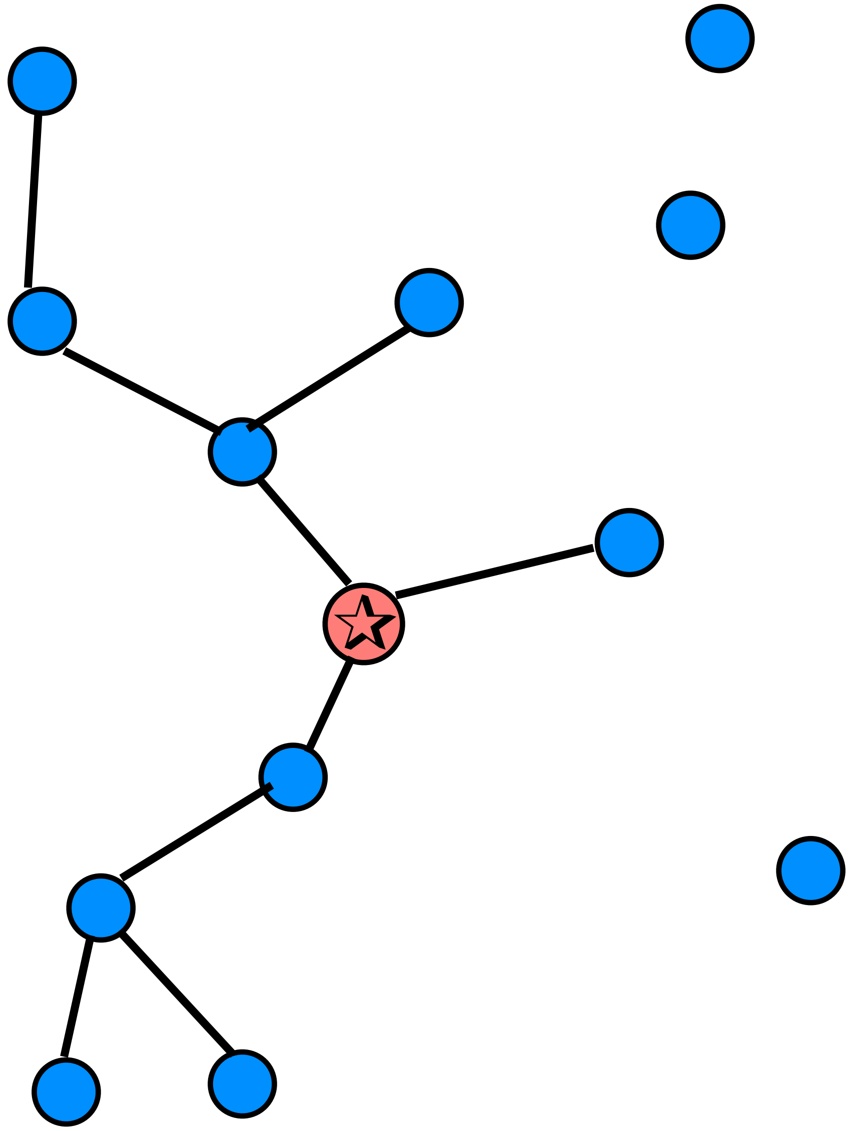
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

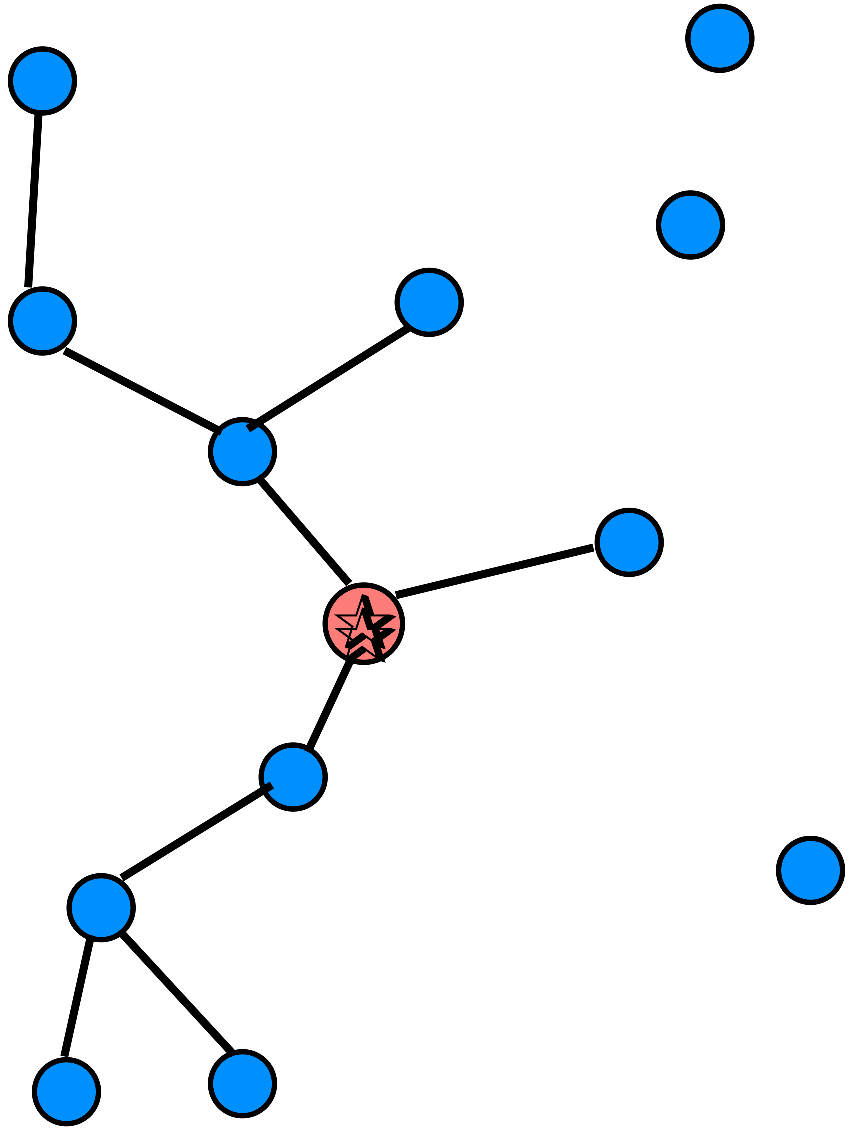
1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Use Case: WSN

- Network consists of a set of **nodes** and one distinguished node **sink**
- Protocol has two phases:

1. **Establishment of a routing tree** (rooted at sink)**:** nodes wirelessly broadcast a special initialization message.
2. **Data collection:** nodes send and forward data via established route using (reliable) unicast messages



Specification in Pwb

Node Behavior

```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
    "init(nodeId)"? (chan) .
    '"init(nodeId)"! <nodeChan> .
    '"data(chan)"<datum> .
    ! "data(nodeChan)"(x).
     '"data(chan)"<x>  ;
```

Specification in Pwb

Node Behavior

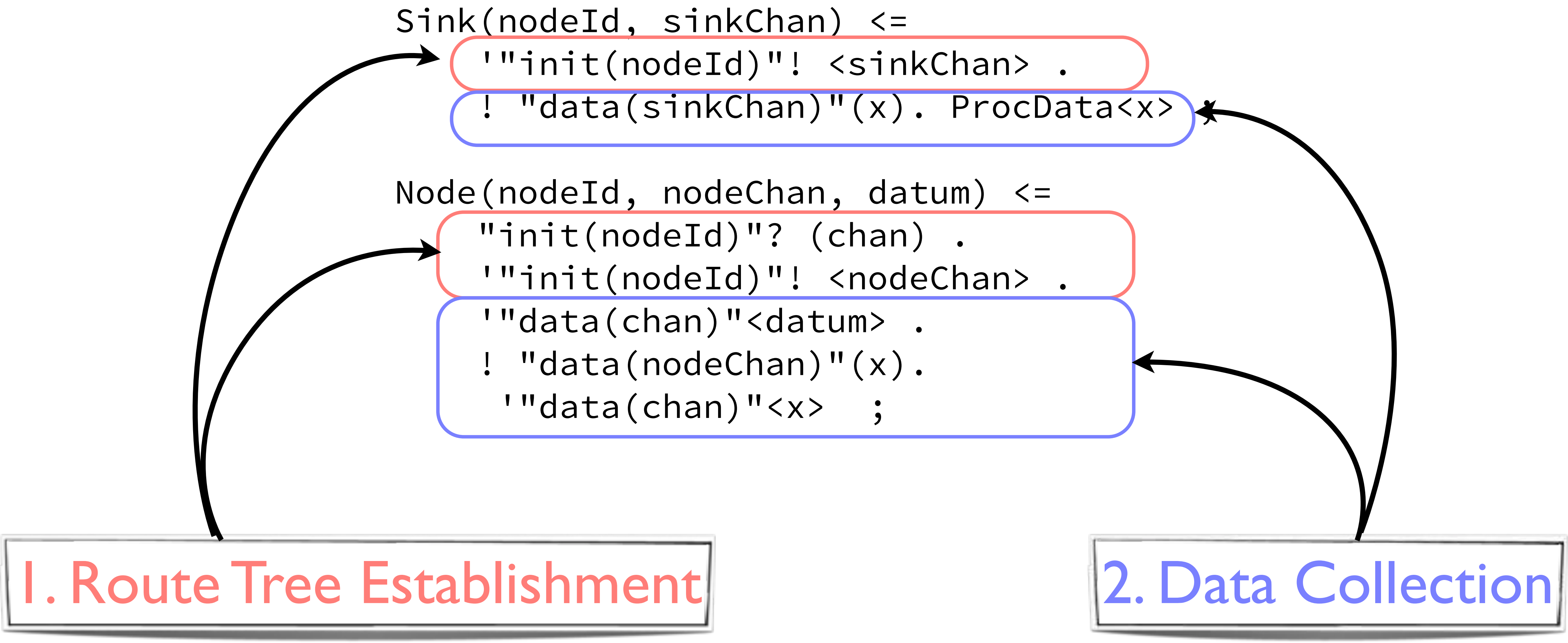
```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
    "init(nodeId)"? (chan) .
    '"init(nodeId)"! <nodeChan> .
    '"data(chan)"<datum> .
    ! "data(nodeChan)"(x).
     '"data(chan)"<x>   ;
```

1. Route Tree Establishment

Specification in Pwb

Node Behavior



```
Sink(nodeId, sinkChan) <=
   '"init(nodeId)"! <sinkChan> .
   ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
   "init(nodeId)"? (chan) .
   '"init(nodeId)"! <nodeChan> .
   '"data(chan)"<datum> .
   ! "data(nodeChan)"(x).
    '"data(chan)"<x>  ;
```

1. Route Tree Establishment

2. Data Collection

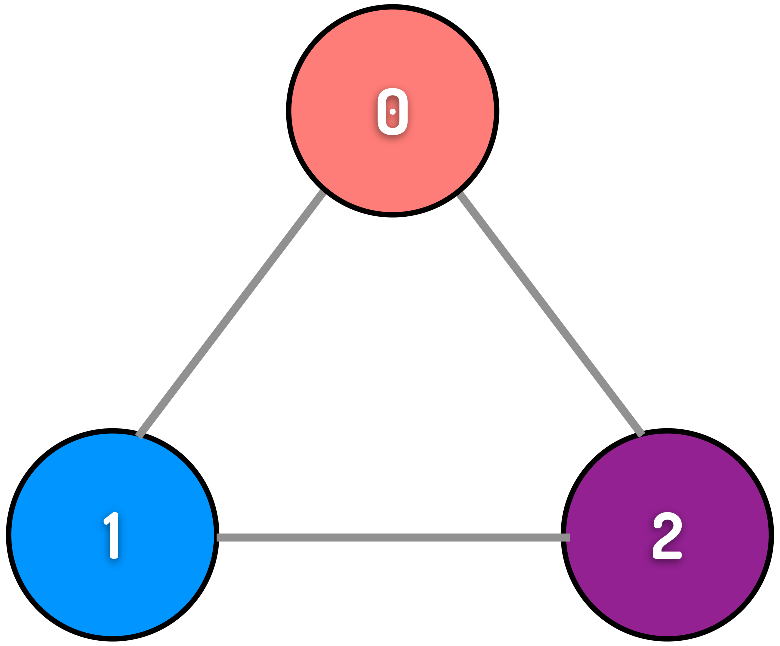
Specification in Pwb

Node Behavior

```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
    "init(nodeId)"? (chan) .
    '"init(nodeId)"! <nodeChan> .
    '"data(chan)"<datum> .
    ! "data(nodeChan)"(x).
     '"data(chan)"<x>  ;
```

Node Connectivity for Broadcasting



graph represented as edge list

```
(0,1), (0,2), (1,2)
```

Specification in Pwb

Node Behavior

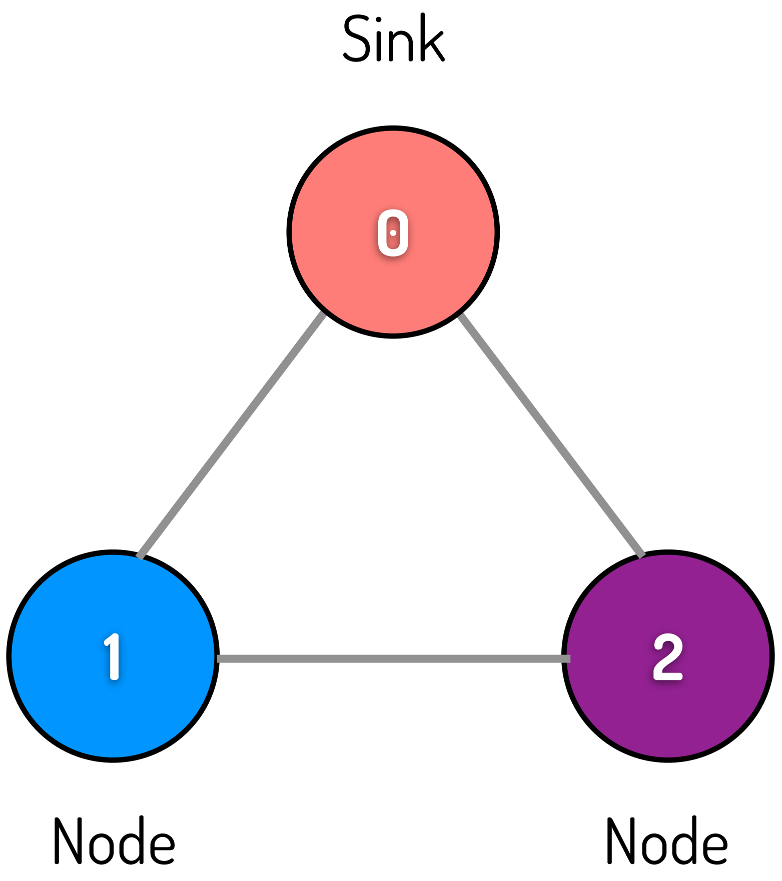
```
Sink(nodeId, sinkChan) <=
    '"init(nodeId)"! <sinkChan> .
    ! "data(sinkChan)"(x). ProcData<x> ;

Node(nodeId, nodeChan, datum) <=
    "init(nodeId)"? (chan) .
    '"init(nodeId)"! <nodeChan> .
    '"data(chan)"<datum> .
    ! "data(nodeChan)"(x).
     '"data(chan)"<x>  ;
```

System

```
(new sinkChan)   Sink<0, sinkChan>           |
(new chan1)      Node<1, chan1, datum1> |
(new chan2)      Node<2, chan2, datum2>
```

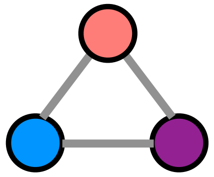
Node Connectivity for Broadcasting



graph represented as edge list

```
(0,1), (0,2), (1,2)
```

Establishment of a Routing Tree (1)

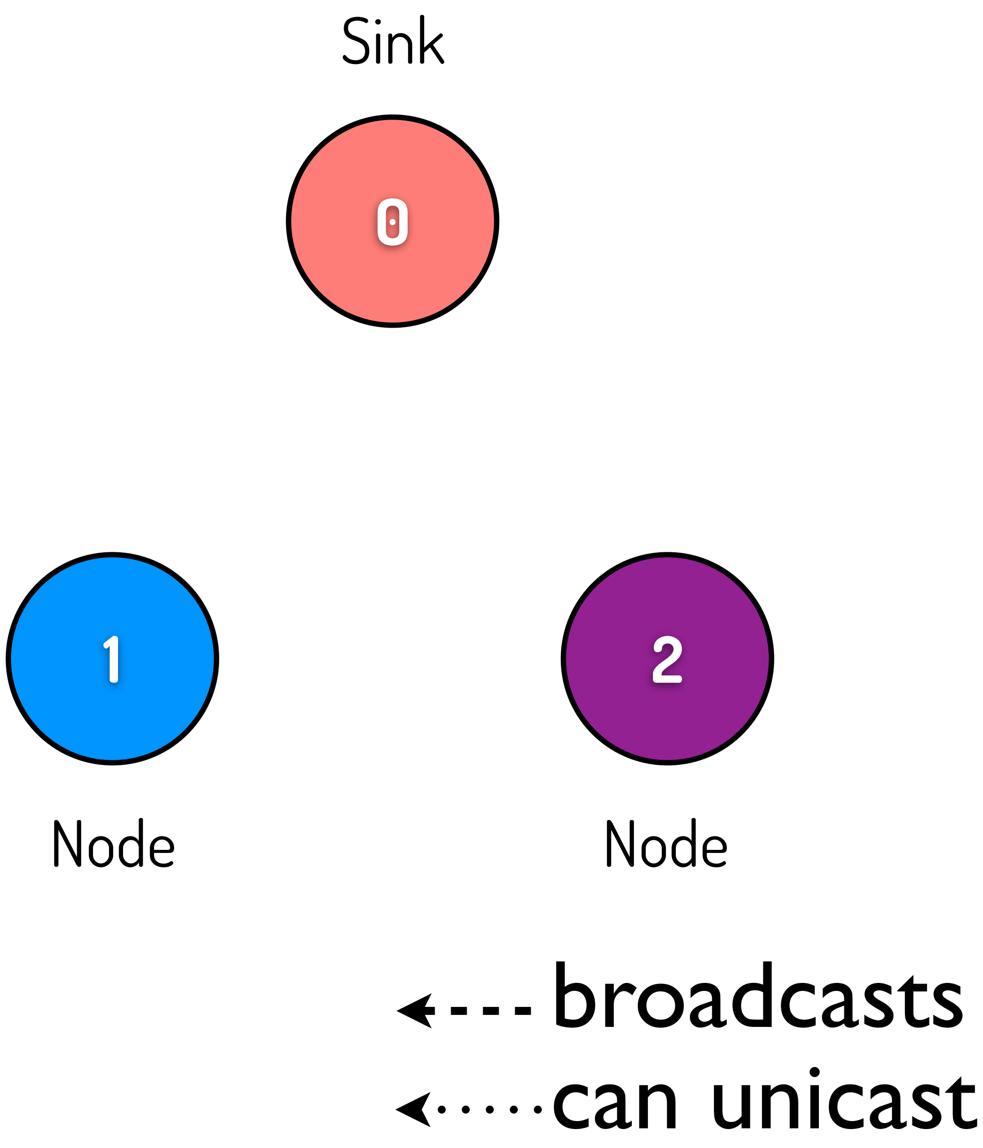


▷

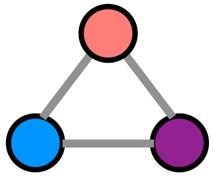
```
(new sinkChan)  Sink<0, sinkChan>      |
(new chan1)     Node<1, chan1, datum1> |
(new chan2)     Node<2, chan2, datum2>



                 true
```



Establishment of a Routing Tree (1)

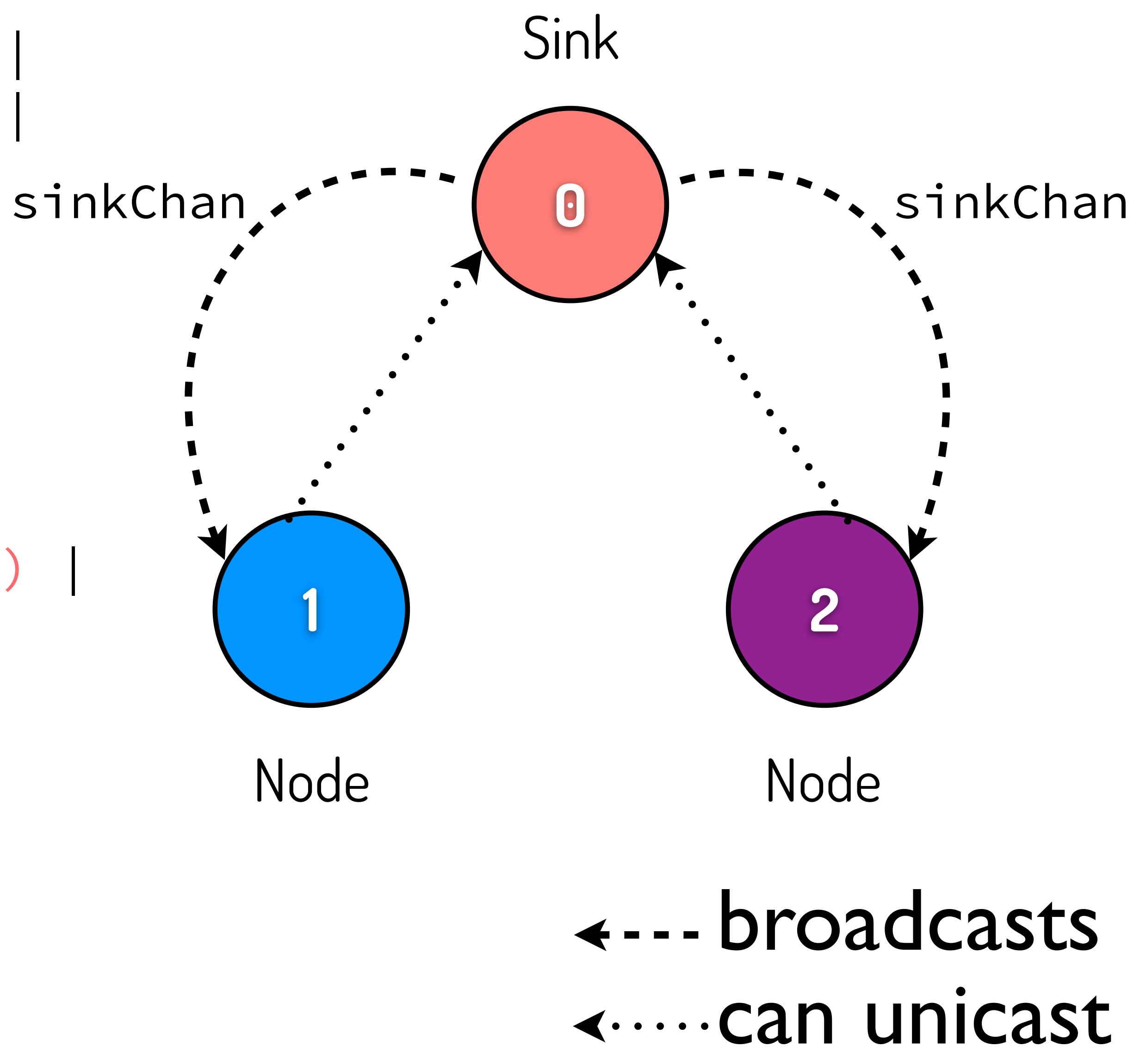


▷

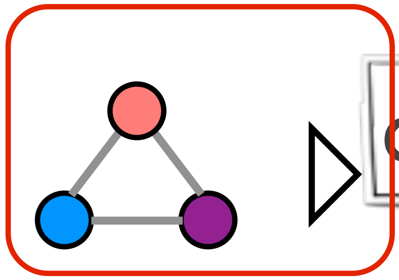
```
(new sinkChan)  Sink<0, sinkChan>      |
(new chan1)     Node<1, chan1, datum1> |
(new chan2)     Node<2, chan2, datum2>
```

```
"init(0)"!(new sinkChan)sinkChan
                true
```

```
(!("data(sinkChan)"(gnb). ProcData<gnb>)) |
  (((new chan1)(
    '"init(1)"!<chan1>.
      '"data(sinkChan)"<datum1>.
        !("data(chan1)"(gnb).
          '"data(sinkChan)"<gnb>))) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gnb).
            '"data(sinkChan)"<gnb>))))
```



Establishment of a Routing Tree (1)

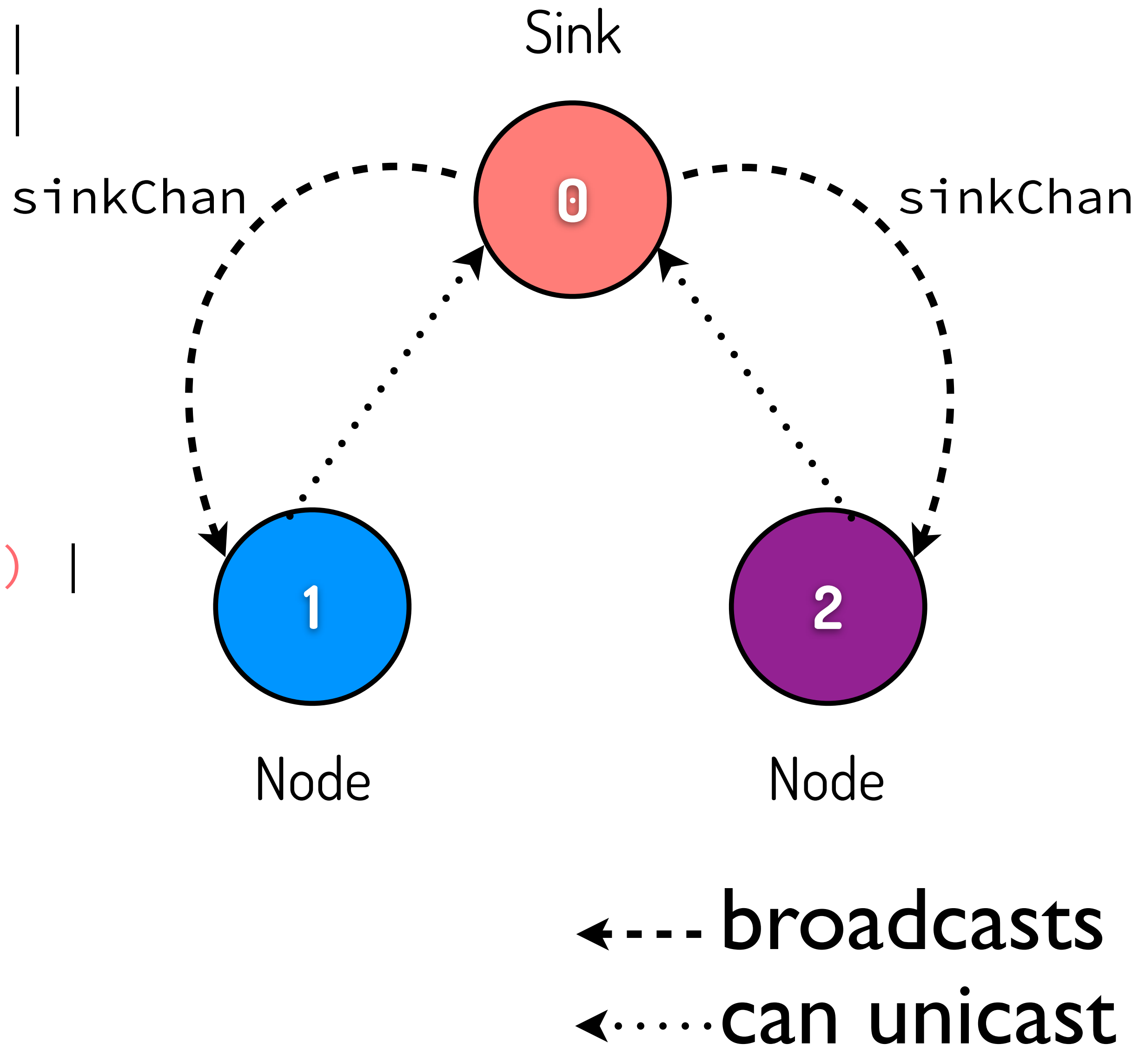


connectivity as current assertion

```
(new sinkChan)  Sink<0, sinkChan>        |
(new chan1)     Node<1, chan1, datum1>   |
(new chan2)     Node<2, chan2, datum2>
```

```
"init(0)"!(new sinkChan)sinkChan
                true
```

```
(!("data(sinkChan)"(gnb). ProcData<gnb>)) |
  (((new chan1)(
    '"init(1)"!<chan1>.
      '"data(sinkChan)"<datum1>.
        !("data(chan1)"(gnb).
          '"data(sinkChan)"<gnb>))) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gnb).
            '"data(sinkChan)"<gnb>))))
```

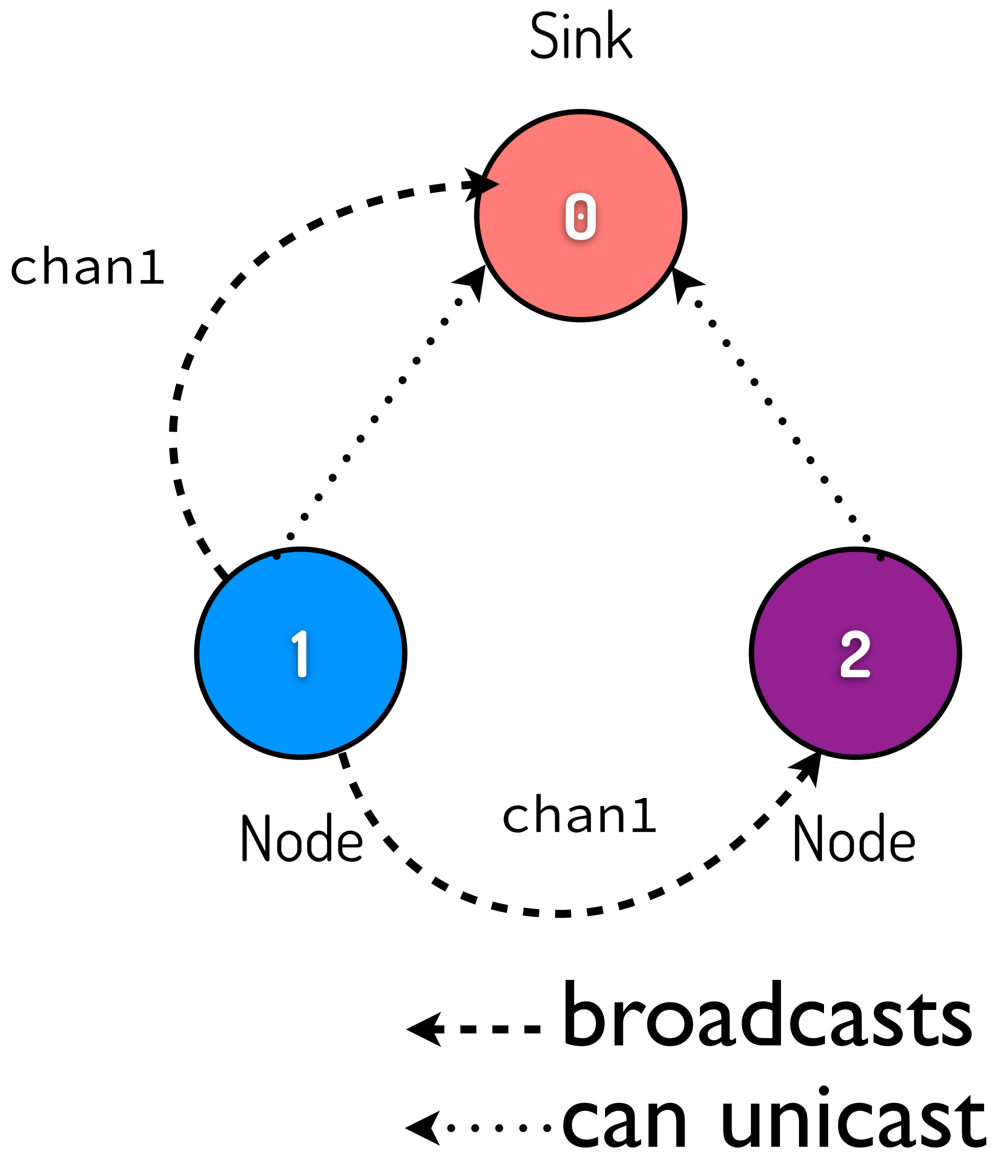


Establishment of a Routing Tree (2)

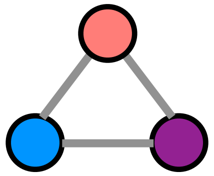
▷

```
(!("data(sinkChan)"(gnb). ProcData<gnb>)) |
  (((new chan1)(
    '"init(1)"!<chan1>.
      '"data(sinkChan)"<datum1>.
        !("data(chan1)"(gnb).
          '"data(sinkChan)"<gnb>))) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gnb).
            '"data(sinkChan)"<gnb>))))
```

```
true
```



Establishment of a Routing Tree (2)

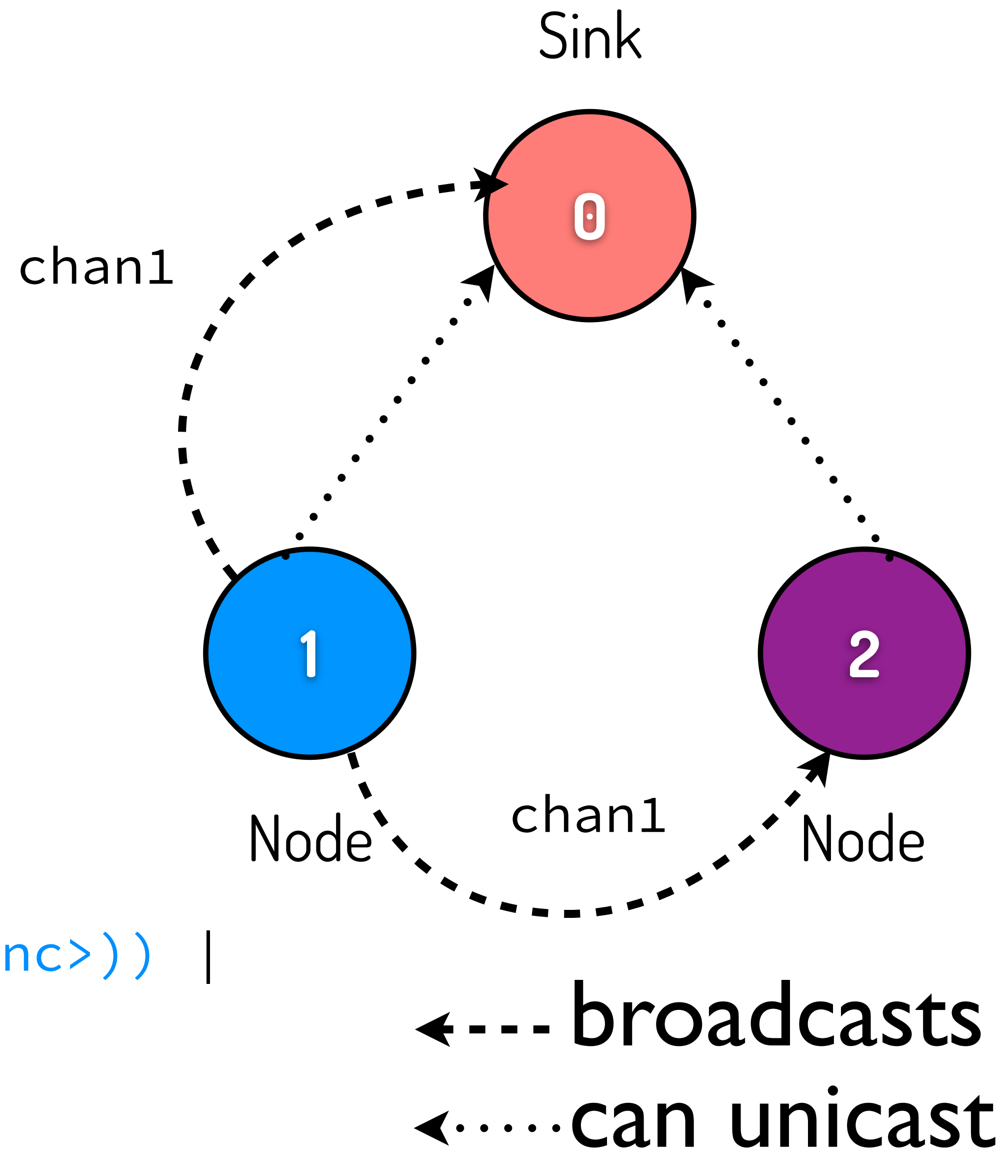


▷

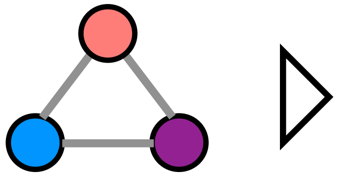
```
(!("data(sinkChan)"(gnb). ProcData<gnb>)) |
 (((new chan1)(
   '"init(1)"!<chan1>.
     '"data(sinkChan)"<datum1>.
       !("data(chan1)"(gnb).
         '"data(sinkChan)"<gnb>))) |
   ((new chan2)(
     '"init(2)"!<chan2>.
       '"data(sinkChan)"<datum2>.
         !("data(chan2)"(gnb).
           '"data(sinkChan)"<gnb>))))
```

```
"init(1)"!(new chan1)chan1
            true
```

```
(!("data(sinkChan)"(gnc). ProcData<gnc>)) |
 (('"data(sinkChan)"<datum1>.
   !("data(chan1)"(gnc). '"data(sinkChan)"<gnc>)) |
   ((new chan2)(
     '"init(2)"!<chan2>.
       '"data(sinkChan)"<datum2>.
         !("data(chan2)"(gnc).
           '"data(sinkChan)"<gnc>))))
```

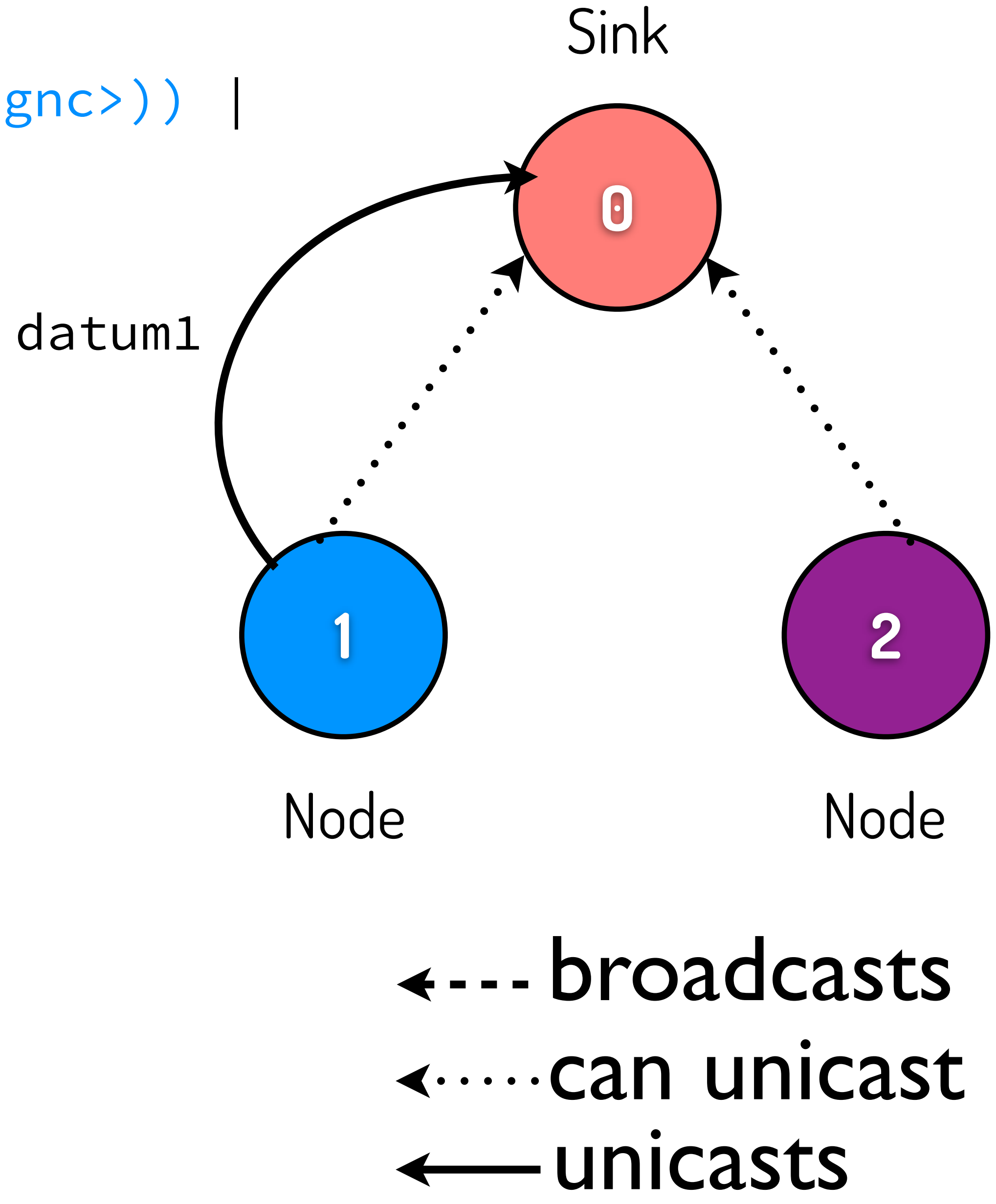


Data Collection

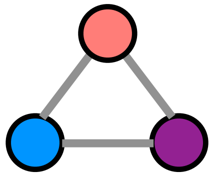


```
(!("data(sinkChan)"(gnc). ProcData<gnc>)) |
  (('"data(sinkChan)"<datum1>.
    !("data(chan1)"(gnc). '"data(sinkChan)"<gnc>)) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gnc).
            '"data(sinkChan)"<gnc>))))
```

true



Data Collection

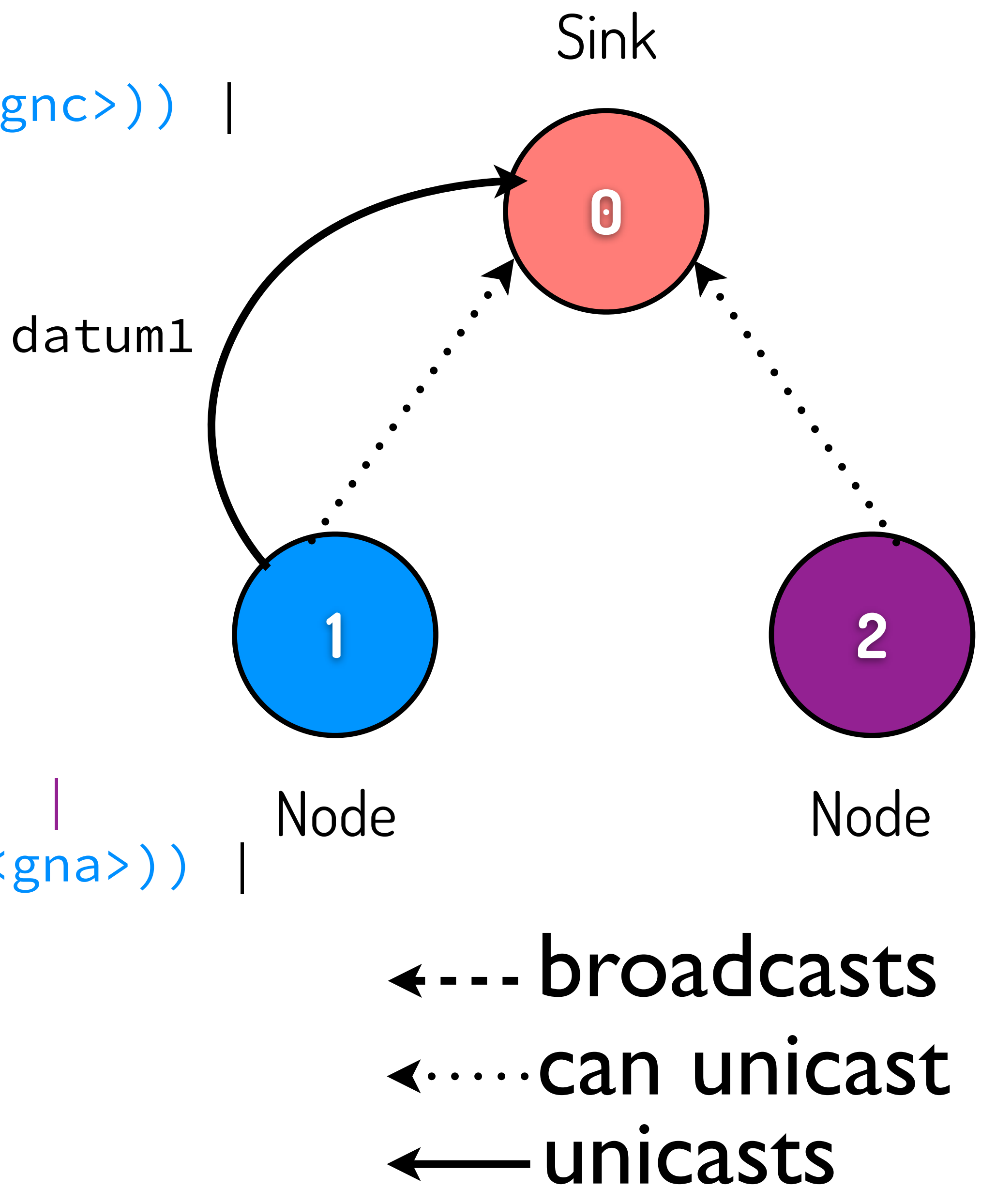


▷

```
(!("data(sinkChan)"(gnc). ProcData<gnc>)) |
  (('"data(sinkChan)"<datum1>.
    !("data(chan1)"(gnc). '"data(sinkChan)"<gnc>)) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gnc).
            '"data(sinkChan)"<gnc>))))
```

tau
true

```
((ProcData<datum1>) |
  (!("data(sinkChan)"(gnb). ProcData<gnb>))) |
  ((!("data(chan1)"(gna). '"data(sinkChan)"<gna>)) |
    ((new chan2)(
      '"init(2)"!<chan2>.
        '"data(sinkChan)"<datum2>.
          !("data(chan2)"(gna).
            '"data(sinkChan)"<gna>))))
```



Weak Bisimulation Checking

```
psi> a(x) ~ *tau*.a(x) ;
([], 1)
```

```
psi> a(x) | b(x) ~
      case T : d(x).b(x) [] T : b(x).a(x);
([d := a], 1)
```

Pwb

User's perspective

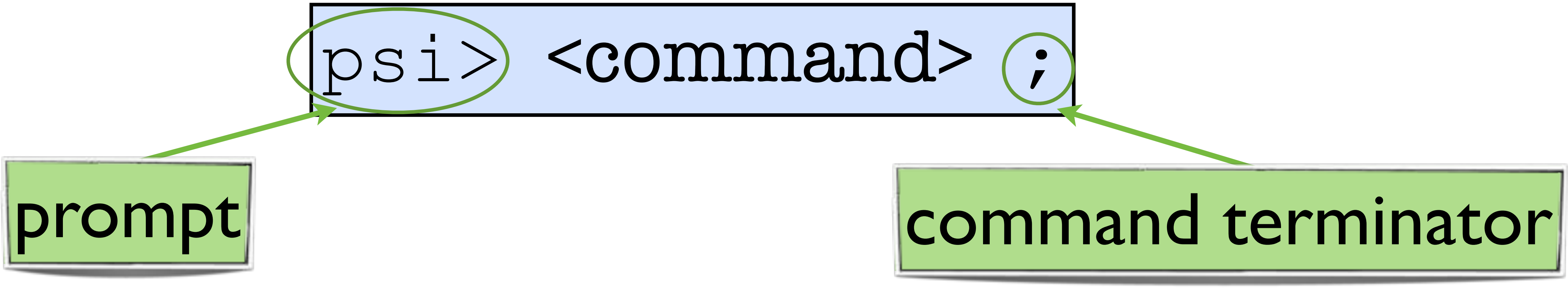
Command Interpreter Syntax Layers

- Commands
- Processes
- Parameters

Commands

```
psi> <command> ;
```

Commands

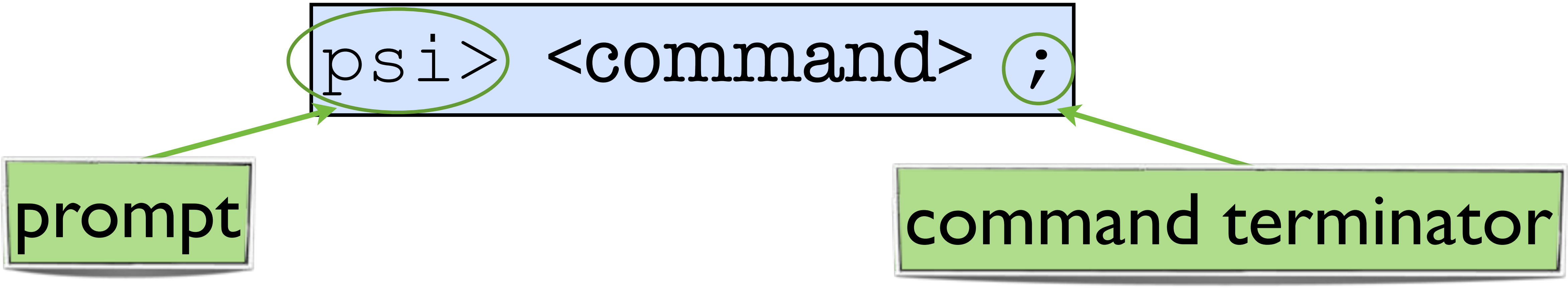


```
psi> <command> ;
```

sstep <process>	symbolic execution interpreter
wsstep <process>	weak symbolic execution
<process> ~ <process>	weak symbolic bisimulation checking
input ”<file name>”	reads commands from file

+ commands altering the process environment

Commands



psi> <command> ;

prompt

command terminator

sstep <process>

wsstep <process>

enter their own command interpreter

[0-9]+ - selecting

q - exiting

b - backtracking

ter

ion

<process> ~ <process> weak symbolic bisimulation checking

input "<file name>" reads commands from file

+ commands altering the processenvironment

P	process
M,N	terms
x	names

<process> Syntax

$M(x_1, \ldots, x_n) . P$	Unicast Input
’ $M <N_1, \ldots, N_n> . P$	Unicast Output
M **?** $(x_1, \ldots, x_n) . P$	Broadcast Input
’ M **!** $<N_1, \ldots, N_n> . P$	Broadcast Output

<process> Syntax

P	process
M,N	terms
x	names

Polyadic

$M(x_1, \ldots, x_n) . P$	Unicast Input
$’\, M \langle N_1, \ldots, N_n \rangle . P$	Unicast Output
$M\, ?\, (x_1, \ldots, x_n) . P$	Broadcast Input
$’\, M\, !\, \langle N_1, \ldots, N_n \rangle . P$	Broadcast Output

P	process
M,N	terms
x	names

<process> Syntax

Polyadic

$M(x_1, \ldots, x_n) . P$	Unicast Input
’ $M \langle N_1, \ldots, N_n \rangle . P$	Unicast Output
$M\,?\,(x_1, \ldots, x_n) . P$	Broadcast Input
’ $M\,!\,\langle N_1, \ldots, N_n \rangle . P$	Broadcast Output

Not patterns :(

P	process
M,N	terms
x	names

<process> Syntax

Polyadic

M(x_1, ..., x_n) . P	Unicast Input
’ M <N_1, ..., N_n> . P	Unicast Output
M ? (x_1, ..., x_n) . P	Broadcast Input
’ M ! <N_1, ..., N_n> . P	Broadcast Output
tau . P	Silent Prefix

Not patterns :(

<process> Syntax

P	process
M,N	terms
x	names

Polyadic

$M(x_1, ..., x_n) . P$	Unicast Input
$' M <N_1, ..., N_n> . P$	Unicast Output
$M ? (x_1, ..., x_n) . P$	Broadcast Input
$' M ! <N_1, ..., N_n> . P$	Broadcast Output
tau . P	Silent Prefix

Useful, e.g., guarding assertions

Not patterns :(

M ? (x_1, ..., x_n) . P	Broadcast In
’ M ! <N_1, ..., N_n> . P	Broadcast C
tau . P	Silent Prefix
case phi_1 : P [] ... [] phi_n : P	Case
(new a) P	Restriction
P | Q	Parallel
(| Psi |)	Assertion
! P	Replication

P	process
M,N	terms
Psi	assertion
phi	condition
x,a	names

$M\ ?\ (x_1, \ldots, x_n)\ .\ P$	Broadcast In
’ $M\ !\ <N_1, \ldots, N_n>\ .\ P$	Broadcast C
$*tau*\ .\ P$	Silent Prefix
case $phi_1 : P\ []\ \ldots\ []\ phi_n : P$	Case
(new a) P	Restriction
$P \mid Q$	Parallel
$(\mid Psi \mid)$	Assertion
! P	Replication
$A< M_1, \ldots, M_n >$	Process Invocation

P	process
M,N	terms
Psi	assertion
phi	condition
x,a	names

M ? (x_1, ..., x_n) . P	Broadcast In
’ M ! <N_1, ..., N_n> . P	Broadcast O
tau . P	Silent Prefix
case phi_1 : P [] ... [] phi_n : P	Case
(new a) P	Restriction
P | Q	Parallel
(| Psi |)	Assertion
! P	Replication
A< M_1, ..., M_n >	Process Invocation

P	process
M,N	terms
Psi	assertion
phi	condition
x,a	names

Similar to HO-Psi’s **run**

$M\ ?\ (x_1, \ldots, x_n)\ .\ P$ Broadcast In

’ $M\ !\ <N_1, \ldots, N_n>\ .\ P$ Broadcast O

$*$tau$*$. P Silent Prefix

P	process
M,N	terms
Psi	assertion
phi	condition
x,a	names

case $\text{phi}_1 : P$ [] … [] $\text{phi}_n : P$

Case

Restriction

Parallel

Assertion

Process constant
==
identifier in a process environment

! P Replication

$\boxed{A}< M_1, \ldots, M_n >$ Process Invocation

Similar to HO-Psi’s **run**

Process Environment

is an environment of processes

Process Environment

is an environment of ~~processes~~ **process clauses**

$$\boxed{A(x_1, \ldots, x_n) <= P}$$

Process Environment

is an environment of ~~processes~~ **process clauses**

$$\boxed{\mathtt{A}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \;\texttt{<=}\; \mathtt{P}}$$

Req. $\mathtt{x}_1, \ldots, \mathtt{x}_n$ must be in the support of $\mathtt{P}$

assertions must be guarded in $\mathtt{P}$

Process Environment

is an environment of ~~processes~~ **process clauses**

$$\boxed{A(x_1, \ldots, x_n) \Leftarrow P}$$

Req. $x_1, \ldots, x_n$ must be in the support of P

assertions must be guarded in P

Ex. ✓ Proc2(chan) <= chan(x).0

✗ Proc1() <= chan(x).0

✓ Proc3(chan) <= chan(x).(| Psi |)

✗ Proc1() <= (| Psi |)

Process Environment

is an environment of ~~processes~~ **process clauses**

$A(x_1, \ldots, x_n) \Leftarrow P$

Req. $x_1, \ldots, x_n$ must be in the support of P

assertions must be guarded in P

Ex. ✓ Proc2(chan) <= chan(x).0

✗ Pro

✓ Proc3(c

✗ Pro

Pwb accepts them by giving a warning, however you won't produce transitions

Process Environment

is an environment of ~~processes~~ **process clauses**

A(x_1, ..., x_n) <= P

Req. x_1, ..., x_n must be in the support of P

nust be guarded in P

also command: inserts clause into env.

han) <= chan(x).0

✘ Pr

✓ Proc3(c

✘ Pro

Pwb accepts them by giving a warning, however you won’t produce transitions

Process Invocation

Non-determinism

```
def {
   Proc(x) <= x(x);
   Proc(x) <= ‘x<x>;
}
```

invocation

`Proc<chan>`

has two transitions

$$\xrightarrow[c]{x(x)} 0$$

$$\xrightarrow[c']{\text{‘x<x>}} 0$$

Mutually Recursive

```
def {
   Proc(x) <= x(x).Agnt<x>;
   Agnt(x) <= ‘x<x>.Proc<x>;
}
```

Cycles are not allowed

```
Proc(x) <= Proc<x>
```

has not transtitions

Parameter Syntax

The params. M, N, phi, Psi
and names x, a

are anything that the implementer intended

however

non alpha-numeric strings need to be **quoted**

M(x_1, ..., x_n) . P

’ M <N_1, ..., N_n> . P

M **?** (x_1, ..., x_n) . P

’ M **!** <N_1, ..., N_n> . P

case phi$_1$: P [] ... [] phi$_n$: P

(new a) P

P | Q

(| Psi |)

! P

A< M_1, ..., M_n >

Parameter Syntax

The params. M, N, phi, Psi
and names x, a

are anything that the implementer intended

however

non alpha-numeric strings need to be **quoted**

ex. ✘ ‘addr:port<cons(1, nil)>.0

✓ ‘ ”addr:port”<”cons(1, nil)”>.0

M(x_1, ..., x_n) . P

’ M <N_1, ..., N_n> . P

M **?** (x_1, ..., x_n) . P

’ M **!** <N_1, ..., N_n> . P

case phi_1 : P [] ... [] phi_n : P

(new a) P

P | Q

(| Psi |)

! P

A< M_1, ..., M_n >

Pwb

Intersection: Semantics

Symbolic Semantics

- More abstract
- No infinite branching
- Sound and complete wrt ordinary semantics

Case for Symbolic

In pi-calculus

$$\frac{}{a(x).P \xrightarrow{ay} P[x := y]}\text{In}$$

Any problems computing this rule?

A Case for Symbolic

In pi-calculus

infinite domain

$$\frac{}{a(x).P \xrightarrow{ay} P[x := y]}\,\text{In}$$

Any problems computing this rule?

Thus
Infinitely many transitions

Late Symbolic Semantics

$$\textsc{In}\ \frac{y\#M,P,x}{\underline{M}(\lambda x)x\,.\,P \xrightarrow[\{\!\!\{\mathbf{1}\vdash M\,\dot{\leftrightarrow}\,y\}\!\!\}]{\underline{y}(x)} P}$$

$$\textsc{Out}\ \frac{y\#M,N,P}{\overline{M}\,N\,.\,P \xrightarrow[\{\!\!\{\mathbf{1}\vdash M\,\dot{\leftrightarrow}\,y\}\!\!\}]{\overline{y}N} P}$$

$$\textsc{Case}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\mathbf{case}\ \widetilde{\varphi}:\widetilde{P} \xrightarrow[C\wedge\{\!\!\{\mathbf{1}\vdash\varphi_i\}\!\!\}]{\alpha} P'}\ \mathrm{subj}(\alpha)\#\varphi_i$$

$$\textsc{Rep}\ \frac{P \mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

$$\textsc{Par}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{bn}(\alpha)\#Q\\ \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Com-New}\ \frac{\begin{array}{c}P \xrightarrow[(\nu\widetilde{c_P})\{\!\!\{\Psi'_P\vdash M_P\,\dot{\leftrightarrow}\,y\}\!\!\}\wedge C_P]{\overline{y}(\nu\widetilde{a})N} P'\\ Q \xrightarrow[(\nu\widetilde{c_Q})\{\!\!\{\Psi'_Q\vdash M_Q\,\dot{\leftrightarrow}\,z\}\!\!\}\wedge C_Q]{\underline{z}(x)} Q'\end{array}}{P \mid Q \xrightarrow[C_{\mathrm{com}}]{\tau} (\nu\widetilde{a})(P' \mid Q'[x:=N])}\ \begin{array}{l}\widetilde{a}\#Q\\ \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Scope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha$$

$$\textsc{Open}\ \frac{P \xrightarrow[C]{\overline{y}(\nu\widetilde{a})N} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\overline{y}(\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{l}b\in\mathsf{n}(N)\\ b\#\widetilde{a},y\end{array}$$

Late Symbolic Semantics

$$\textsc{In}\ \frac{y\#M,P,x}{\underline{M}(\lambda x)x\,.\,P \xrightarrow[\{\mathbf{1}\vdash M\dot{\leftrightarrow}y\}]{\underline{y}(x)} P} \qquad \textsc{Out}\ \frac{y\#M,N,P}{\overline{M}\,N\,.\,P \xrightarrow[\{\mathbf{1}\vdash M\dot{\leftrightarrow}y\}]{\overline{y}N} P}$$

$$\textsc{Case}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\mathbf{case}\ \widetilde{\varphi}:\widetilde{P} \xrightarrow[C\wedge\{\mathbf{1}\vdash\varphi_i\}]{\alpha} P'}\ \mathrm{subj}(\alpha)\#\varphi_i \qquad \textsc{Rep}\ \frac{P \mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

$$\textsc{Par}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{bn}(\alpha)\#Q\\ \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Case}\ \frac{\qquad C \qquad}{\textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C \wedge \{\mathbf{1} \vdash \varphi_i\}]{\alpha} P'}\ \mathrm{subj}(\alpha)\#\varphi_i \qquad \textsc{Rep}\ \frac{\qquad C \qquad}{!P \xrightarrow[C]{\alpha} P'}$$

Late Symbolic Semantics

$$\textsc{Par}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q) \otimes C]{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{bn}(\alpha)\#Q \\ \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Com-New}\ \frac{\begin{array}{c} P \xrightarrow[(\nu\widetilde{c_P})\{\Psi'_P \vdash M_P \dot\leftrightarrow y\} \wedge C_P]{\overline{y}(\nu\widetilde{a})N} P' \\ Q \xrightarrow[(\nu\widetilde{c_Q})\{\Psi'_Q \vdash M_Q \dot\leftrightarrow z\} \wedge C_Q]{\underline{z}(x)} Q' \end{array}}{P \mid Q \xrightarrow[C_{\mathrm{com}}]{\tau} (\nu\widetilde{a})(P' \mid Q'[x := N])}\ \begin{array}{l}\widetilde{a}\#Q \\ \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Ope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha \qquad \textsc{Open}\ \frac{P \xrightarrow[C]{\overline{y}(\nu\widetilde{a})N} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\overline{y}(\nu\widetilde{a} \cup \{b\})N} P'}\ \begin{array}{l} b \in \mathsf{n}(N \\ b\#\widetilde{a}, y\end{array}$$

$$C_{\mathrm{com}} =$$

$$((\nu\widetilde{c_P}\widetilde{c_Q})\{\Psi'_P \otimes \Psi'_Q \vdash M_P \dot\leftrightarrow M_Q\}) \wedge (((\nu\widetilde{c_Q})\Psi'_Q) \otimes C_P) \wedge (((\nu\widetilde{c_P})\Psi'_P) \otimes C_Q)$$

composes the frame with each conjunct

Transition Constraints and Solutions

$$
\begin{array}{lll}
 & \textit{Constraint} & \textit{Solutions} \\
C, C' ::= & \mathbf{true} & \{(\sigma, \Psi) \;:\; \sigma \textit{ is a subst. sequence } \wedge \; \Psi \in \mathbf{A}\} \\
 & \mathbf{false} & \emptyset \\
 & (\nu a)C & \{(\sigma, \Psi) \;:\; b\#\sigma, \Psi \wedge (\sigma, \Psi) \in \mathrm{sol}((a\ b) \cdot C)\} \\
 & \{\Psi' \vdash \varphi\} & \{(\sigma, \Psi) \;:\; \Psi'\sigma \otimes \Psi \vdash \varphi\sigma\} \\
 & C \wedge C' & \mathrm{sol}(C) \cap \mathrm{sol}(C')
\end{array}
$$

Solution is a set of substitution and assertion pairs

Transition Constraints and Solutions

$$
\begin{array}{lll}
 & \textit{Constraint} & \textit{Solutions} \\
C, C' ::= & \mathbf{true} & \{(\sigma, \Psi) \ : \ \sigma \ \textit{is a subst. sequence} \ \wedge \ \Psi \in \mathbf{A}\} \\
 & \mathbf{false} & \emptyset \\
 & (\nu a)C & \{(\sigma, \Psi) \ : \ b\#\sigma, \Psi \wedge (\sigma, \Psi) \in \mathrm{sol}((a\ b) \cdot C)\} \\
 & \{\Psi' \vdash \varphi\} & \{(\sigma, \Psi) \ : \ \Psi'\sigma \otimes \Psi \vdash \varphi\sigma\} \\
 & C \wedge C' & \mathrm{sol}(C) \cap \mathrm{sol}(C')
\end{array}
$$

Solution is a set of substitution and assertion pairs

ex. $\underline{\mathsf{nextFreq}(f)}(x).x(y).0 \mid \overline{\mathsf{nextFreq}(g)}\langle \mathsf{nextFreq}(\mathsf{a})\rangle.0$

$$\xrightarrow[\{\mathsf{nextFreq}(f) \dot{\leftrightarrow} \mathsf{nextFreq}(g)\}]{\tau} \underline{\mathsf{nextFreq}(\mathsf{a})}(y).0$$

Transition Constraints and Solutions

	Constraint	*Solutions*
$C, C' ::=$	$\mathbf{true}$	$\{(\sigma, \Psi) \;:\; \sigma \textit{ is a subst. sequence } \wedge \; \Psi \in \mathbf{A}\}$
	$\mathbf{false}$	$\emptyset$
	$(\nu a)C$	$\{(\sigma, \Psi) \;:\; b\#\sigma, \Psi \wedge (\sigma, \Psi) \in \mathrm{sol}((a\ b) \cdot C)\}$
	$\{\Psi' \vdash \varphi\}$	$\{(\sigma, \Psi) \;:\; \Psi'\sigma \otimes \Psi \vdash \varphi\sigma\}$
	$C \wedge C'$	$\mathrm{sol}(C) \cap \mathrm{sol}(C')$

Solution is a set of substitution and assertion pairs

ex. $\underline{\mathsf{nextFreq}(f)}(x).x(y).0 \mid \overline{\mathsf{nextFreq}(g)}\langle\mathsf{nextFreq}(\mathsf{a})\rangle.0$

$$\xrightarrow[\{\mathsf{nextFreq}(f)\dot{\leftrightarrow}\mathsf{nextFreq}(g)\}]{\tau} \underline{\mathsf{nextFreq}(\mathsf{a})}(y).0$$

$$\mathrm{sol}(\{\mathsf{nextFreq}(f)\dot{\leftrightarrow}\mathsf{nextFreq}(g)\}) = \{([f := g], 1), ([g := f], 1), \ldots$$

Transition Constraints and Solutions

$$\begin{array}{lll}
 & & \textit{Constraint Solutions} \\
C, C' ::= & \mathbf{true} & \{(\sigma, \Psi) \ : \ \sigma \ \textit{is a subst. sequence} \ \wedge \ \Psi \in \mathbf{A}\} \\
 & \mathbf{false} & \emptyset \\
 & (\nu a)C & \{(\sigma, \Psi) \ : \ b\#\sigma, \Psi \wedge (\sigma, \Psi) \in \mathrm{sol}((a\ b) \cdot C)\} \\
 & \{\!\!\{\Psi' \vdash \varphi\}\!\!\} & \{(\sigma, \Psi) \ : \ \Psi'\sigma \otimes \Psi \vdash \varphi\sigma\} \\
 & C \wedge C' & \mathrm{sol}(C) \cap \mathrm{sol}(C')
\end{array}$$

Solution is a set of substitution and assertion pairs

ex. $\underline{\mathsf{nextFreq}(f)}(x).x(y).0 \mid \overline{\mathsf{nextFreq}(g)}\langle\mathsf{nextFreq(a)}\rangle.0$

$$\xrightarrow[\{\!\!\{\mathsf{nextFreq}(f)\dot{\leftrightarrow}\mathsf{nextFreq}(g)\}\!\!\}]{\tau} \ \underline{\mathsf{nextFreq(a)}}(y).0$$

$$\mathrm{sol}(\{\!\!\{\mathsf{nextFreq}(f)\dot{\leftrightarrow}\mathsf{nextFreq}(g)\}\!\!\}) = \{([f := g], 1), ([g := f], 1), \ldots$$

Finding a solution ~ solving sat. problem

Bisim Constraints

	Constraint	*The solutions* $\mathrm{sol}(C)$ *are all pairs* (σ, Ψ) *such that*
$C, C' ::=$	C_t	$(\sigma, \Psi) \models C_t$
	$\{\!\mid M = N \mid\!\}$	$M\sigma = N\sigma$
	$\{\!\mid a \# X \mid\!\}$	$(a \# X)\sigma$ *and* $a \# dom(\sigma)$
	$C \wedge C'$	$(\sigma, \Psi) \models C$ *and* $(\sigma, \Psi) \models C'$
	$C \vee C'$	$(\sigma, \Psi) \models C$ *or* $(\sigma, \Psi) \models C'$
	$C \Rightarrow C'$	$\forall \Psi'.(\sigma, \Psi \otimes \Psi') \models C$ *implies* $(\sigma, \Psi \otimes \Psi') \models C'$
	$\forall x.C$	$\bigcap_{M \in \mathbf{T}} sol(C[x := M])$

Pwb

Implementer’s perspective

Instance Parameters

Definition 1 (Psi-calculus parameters)**.** *A psi-calculus requires the three (not necessarily disjoint) nominal data types:*

$\mathbf{T}$ *the (data) terms, ranged over by* M, N
$\mathbf{C}$ *the conditions, ranged over by* φ
$\mathbf{A}$ *the assertions, ranged over by* Ψ

and the four equivariant operators:

$\dot{\leftrightarrow} : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$ *Channel Equivalence*
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$ *Composition*
$\mathbf{1} : \mathbf{A}$ *Unit*
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$ *Entailment*

and substitution functions $[\tilde{a} := \tilde{M}]$*, substituting terms for names, on all of* $\mathbf{T}$*,* $\mathbf{C}$*, and* $\mathbf{A}$*.*

Instance Requisites

Channel symmetry: $\Psi \vdash M \dot{\leftrightarrow} N \implies \Psi \vdash N \dot{\leftrightarrow} M$

Channel transitivity: $\Psi \vdash M \dot{\leftrightarrow} N \;\wedge\; \Psi \vdash N \dot{\leftrightarrow} L \implies \Psi \vdash M \dot{\leftrightarrow} L$

Composition: $\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$

Identity: $\Psi \otimes \mathbf{1} \simeq \Psi$

Associativity: $(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$

Commutativity: $\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$

Weakening: $\Psi \vdash \varphi \implies \Psi \otimes \Psi' \vdash \varphi$

Names are terms: $\mathcal{N} \subseteq \mathbf{T}$

Instance Requisites

Channel symmetry: $\Psi \vdash M \dot\leftrightarrow N \implies \Psi \vdash N \dot\leftrightarrow M$

Channel transitivity: $\Psi \vdash M \dot\leftrightarrow N \;\wedge\; \Psi \vdash N \dot\leftrightarrow L \implies \Psi \vdash M \dot\leftrightarrow L$

Composition: $\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$

Identity: $\Psi \otimes \mathbf{1} \simeq \Psi$

Associativity: $(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$

Commutativity: $\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$

Weakening: $\Psi \vdash \varphi \implies \Psi \otimes \Psi' \vdash \varphi$

Names are terms: $\mathcal{N} \subseteq \mathbf{T}$

New

Instance Requisites

Channel symmetry:	$\Psi \vdash M \dot{\leftrightarrow} N \implies \Psi \vdash N \dot{\leftrightarrow} M$
Channel transitivity:	$\Psi \vdash M \dot{\leftrightarrow} N \;\wedge\; \Psi \vdash N \dot{\leftrightarrow} L \implies \Psi \vdash M \dot{\leftrightarrow} L$
Composition:	$\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$
Identity:	$\Psi \otimes \mathbf{1} \simeq \Psi$
Associativity:	$(\Psi \otimes \Psi') \otimes \Psi$
Commutativity:	$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$
Weakening:	$\Psi \vdash \varphi \implies \Psi \otimes \Psi' \vdash \varphi$
Names are terms:	$\mathcal{N} \subseteq \mathbf{T}$

Only for bisimulation alg.

New

+ Substitution

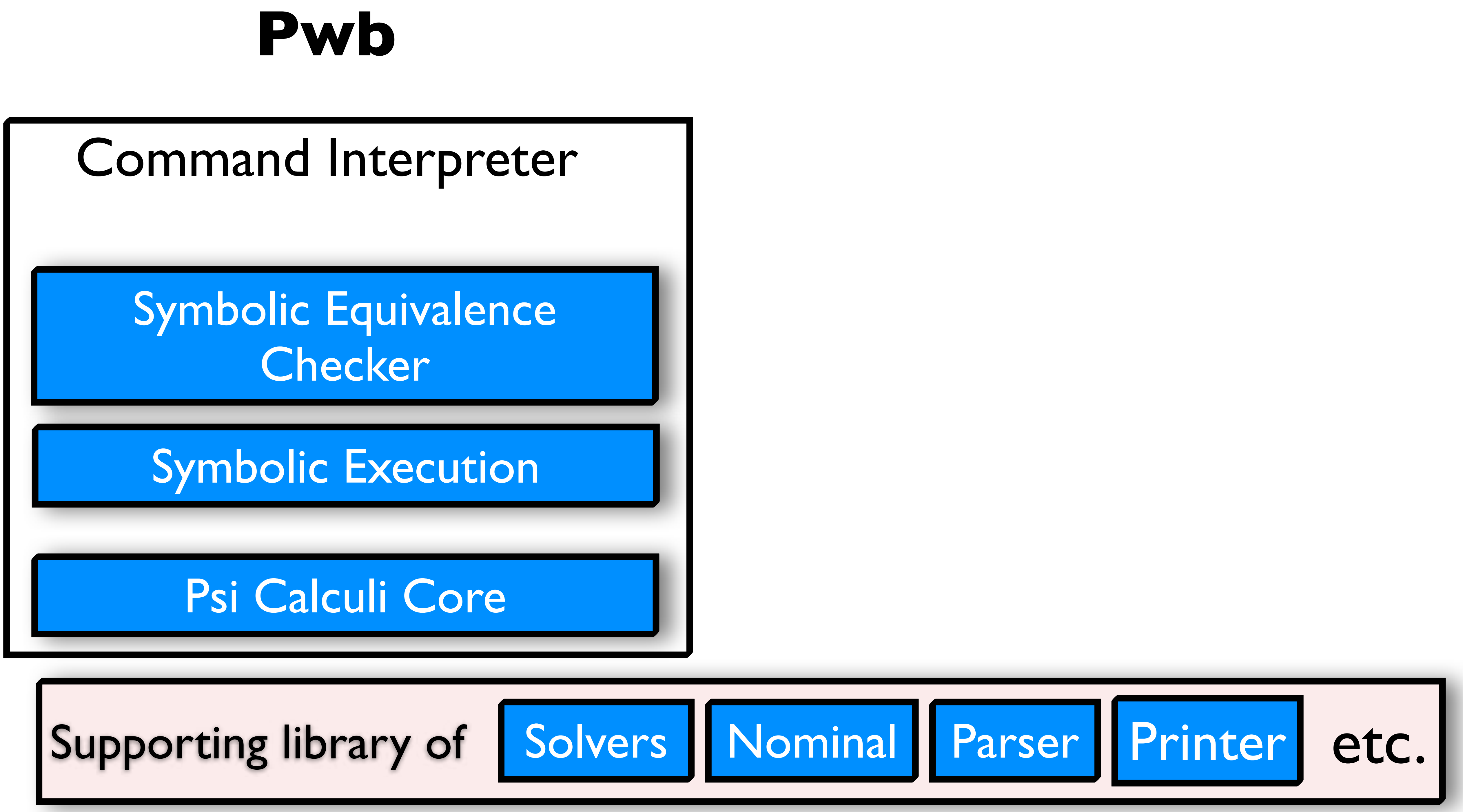
$$Substitution:$$

$$(\forall X \in \{\mathbf{T}, \mathbf{A}, \mathbf{C}\}) \quad \tilde{b}\#X, \tilde{a} \implies X[\tilde{a} := \tilde{M}] = ((\tilde{a}\ \tilde{b}) \cdot X)[\tilde{b} := \tilde{M}]$$

Can lose names!

$$\begin{array}{rcl}
X[x := x] & = & X \\
x[x := M] & = & M \\
X[x := M] & = & X \text{ if } x\#X \\
X[x := L][y := M] & = & X[y := M][x := L] \text{ if } x\#y, M \text{ and } y\#L
\end{array}$$

Architecture



Pwb

Command Interpreter

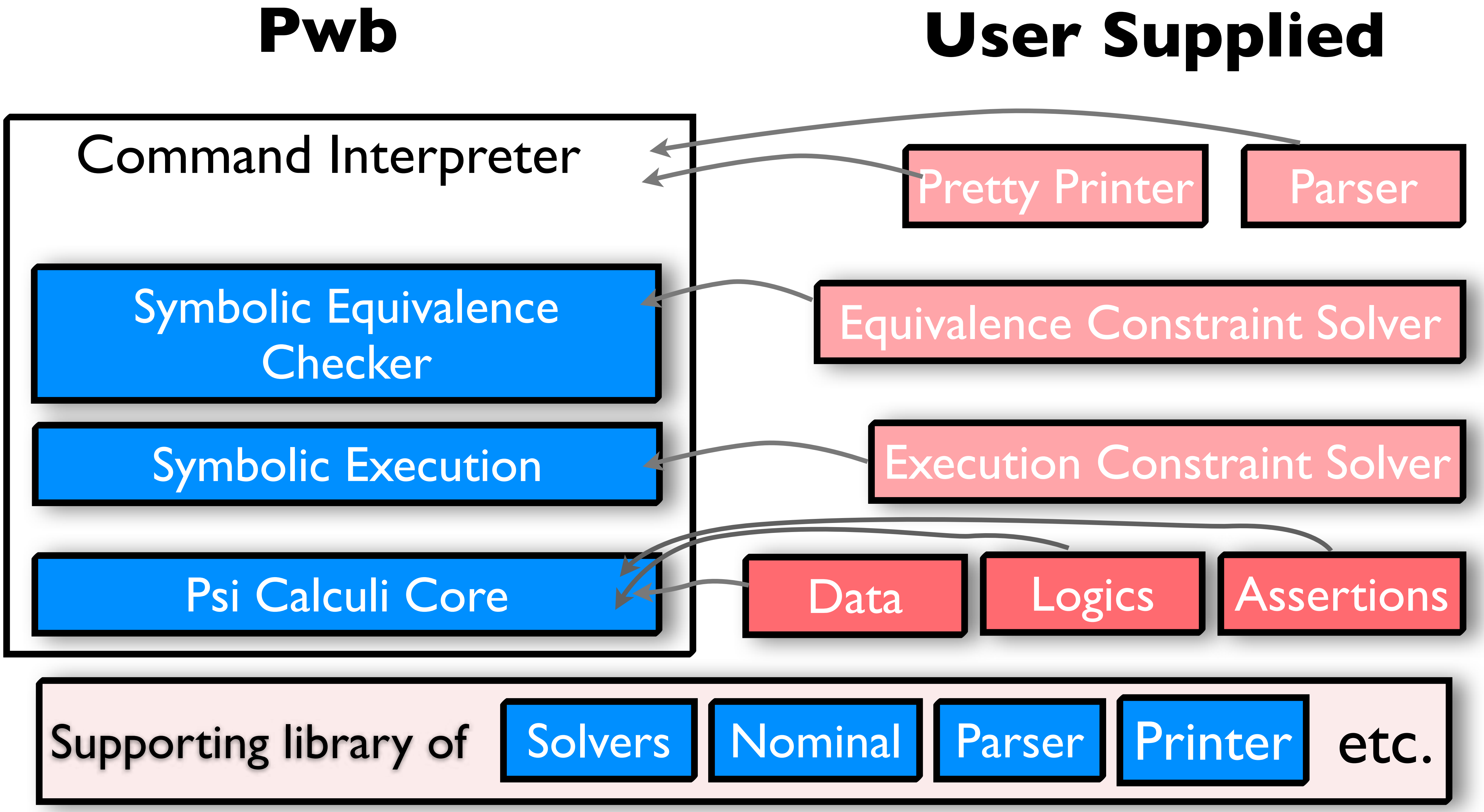
Symbolic Equivalence Checker

Symbolic Execution

Psi Calculi Core

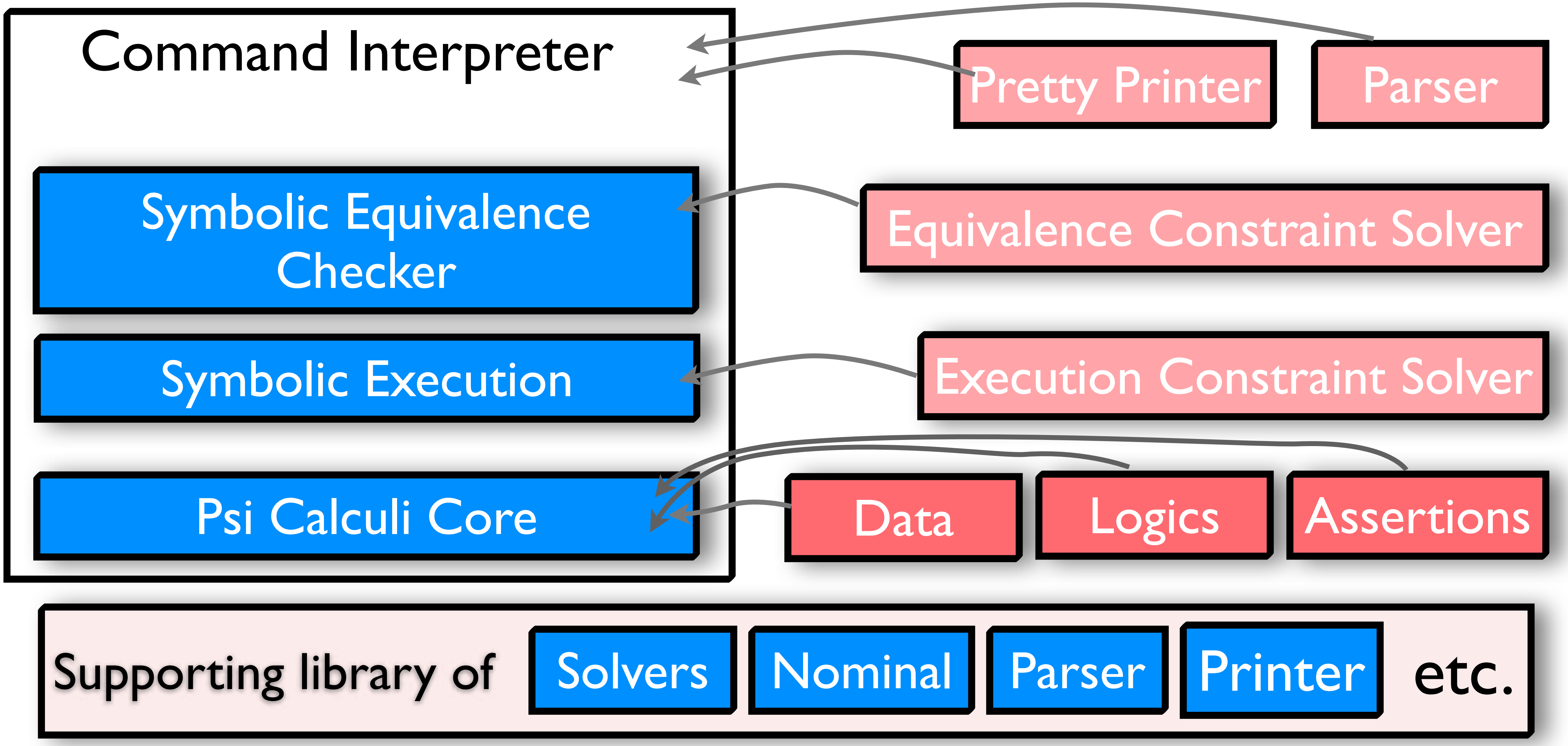
Supporting library of Solvers Nominal Parser Printer etc.

Architecture



Architecture

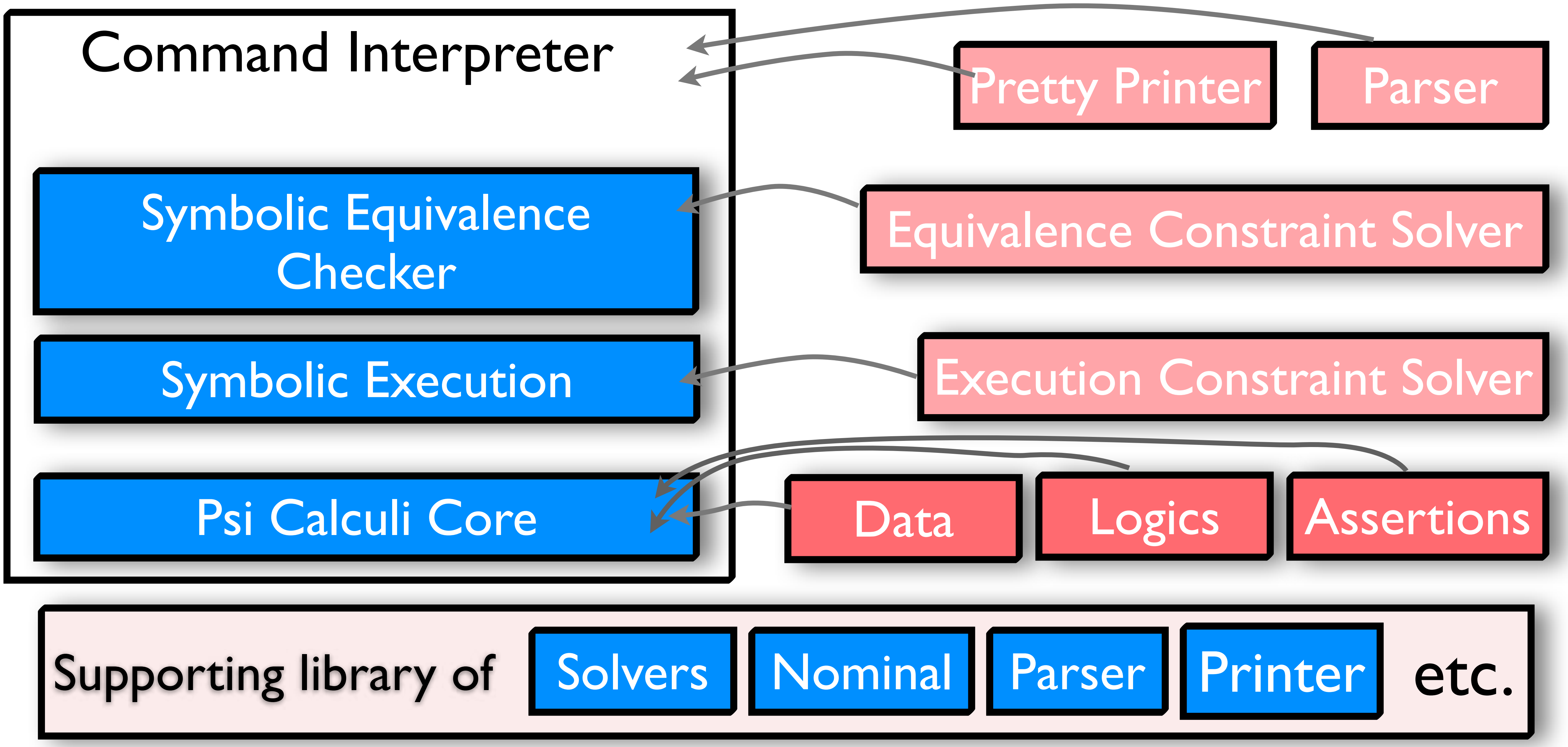
```
$ pwb load-instance <instance>.ML
```



Architecture

Standard ML

```
$ pwb load-instance <instance>.ML
```



Command Interpreter

Symbolic Equivalence Checker

Symbolic Execution

Psi Calculi Core

Pretty Printer

Parser

Equivalence Constraint Solver

Execution Constraint Solver

Data

Logics

Assertions

Supporting library of Solvers Nominal Parser Printer etc.

Included Constraint Solvers

Simple SMT

```
signature PWB_SMT_THEORY =
sig
  type literal
  val neg : literal -> literal
  val eqL : literal -> literal -> bool

  type model
  val empty : model
  val extend  : model -> literal -> model
  val forget : model -> literal list -> model
  val isConsistent : model -> PwbSMTTypes.strength -> bool
  val models : model -> literal -> bool
end;
```

ILP

```
signature ILP =
sig
  type var = string
  datatype rel = Eq | Lt | Gt | LtE | GtE
  type equation =
    ((int * string) list) * rel * ((int * string) list)
  type equation_system= equation list
  type solution = (var * int) list
  val solve : equation_system -> (string, solution) Either.eit
end;
```

Example Implementation

$$
\begin{aligned}
\mathbf{T} &\overset{\text{def}}{=} \mathcal{N} \cup \{\mathsf{nextFreq}(M) : M \in \mathbf{T}\} \\
\mathbf{C} &\overset{\text{def}}{=} \{M = N : M, N \in \mathbf{T}\} \cup \{\top\} \\
\mathbf{A} &\overset{\text{def}}{=} \{1\} \\
\mathbf{1} &\overset{\text{def}}{=} 1 \\
\dot{\leftrightarrow} &\overset{\text{def}}{=} = \\
\otimes &\overset{\text{def}}{=} \lambda\langle \Psi_1, \Psi_2 \rangle.1 \\
\vdash &\overset{\text{def}}{=} \{\langle 1, M = M \rangle : M \in \mathbf{T}\} \cup \{\langle 1, \top \rangle\}
\end{aligned}
$$

Ex.

$$
\underline{\mathsf{nextFreq}(x)}(f).P \ \mid \ \overline{\mathsf{nextFreq}(x)}\langle \mathsf{nextFreq}(y) \rangle \xrightarrow{\tau} P[f := \mathsf{nextFreq}(y)] \mid \mathbf{0}
$$

Parameters

```
type     name      = string

datatype term = Name     of name
              | NextFreq of term

datatype condition = Eq of term * term | True
datatype assertion = Unit

val unit = Unit
val chaneq = Eq
fun compose _ = Unit

fun entails (Unit,Eq (m, n)) = (m = n)
  | entails (Unit,True)      = true

fun var a = Name a
```

Not really required

Substitution

```
fun substT sigma (Name a)    =
      (case List.find (fn (b,_) => a = b) sigma of
           NONE       => Name a
         | SOME (_,t) => t)
  | substT sigma (NextFreq n) = NextFreq (substT sigma n)

fun substC s True           = True
  | substC s (Eq (t1, t2)) = Eq (substT s t1, substT s t2)

fun substA _ _ = Unit
```

Nominal

```
fun new xvec          = StringName.generateDistinct xvec
fun newBasedOn _ xvec = new xvec
fun swap_name (a,b) n = StringName.swap_name (a,b) n

fun supportT (Name n)     = [n]
  | supportT (NextFreq m) = supportT m
fun supportC (Eq (m, n))  = supportT m @ supportT n
  | supportC True         = []
fun supportA _            = []

fun swapT pi (Name n)      = Name (swap_name pi n)
  | swapT pi (NextFreq t)  = NextFreq (swapT pi t)
fun swapC _  True          = True
  | swapC pi (Eq (t1, t2)) = Eq (swapT pi t1, swapT pi t2)
fun swapA _ _ = Unit

fun eqT _ (a,b) = a = b
fun eqC _ (a,b) = a = b
fun eqA _ (a,b) = a = b
```

Constraint Solving

$$(\nu\widetilde{a})\{\!|\mathrm{nextFreq}(N) \mathrel{\dot{\leftrightarrow}} \mathrm{nextFreq}(M)|\!\} \land C \rightarrowtail (\nu\widetilde{a})\{\!|N \mathrel{\dot{\leftrightarrow}} M|\!\} \land C \tag{\textsc{Decom}}$$

$$(\nu\widetilde{a})\{\!|\mathrm{nextFreq}(N) \mathrel{\dot{\leftrightarrow}} a|\!\} \land C \rightarrowtail (\nu\widetilde{a})\{\!|a \mathrel{\dot{\leftrightarrow}} \mathrm{nextFreq}(N)|\!\} \land C \tag{\textsc{Swap}}$$

$$(\nu\widetilde{a})\{\!|\top|\!\} \land C \rightarrowtail C \tag{\textsc{TrT}}$$

$$(\nu\widetilde{a})\{\!|a \mathrel{\dot{\leftrightarrow}} a|\!\} \land C \rightarrowtail C \tag{\textsc{TrEq}}$$

$$(\nu\widetilde{a})\{\!|a \mathrel{\dot{\leftrightarrow}} N|\!\} \land C \overset{[a:=N]}{\rightarrowtail} C[a := N] \qquad \text{if } a, N \# \widetilde{a} \ \land \ a \# N \tag{\textsc{Elim}}$$

$$(\nu\widetilde{a})\{\!|a \mathrel{\dot{\leftrightarrow}} N|\!\} \land C \rightarrowtail \blacksquare \qquad \text{if } a \neq N \ \land \ (a \in n(N) \lor a \in \widetilde{a} \lor n(N) \subseteq \widetilde{a}) \tag{\textsc{Fail}}$$

Constraint Solver

```
                   (composeSubst sigma (a, n))
             else
               Either.LEFT [(Eq (Name a, n))]
    | mgu _ _ = Err.error "explode failed in fhss.ML"

fun explode (avec,psi,phis) = map (fn phi => (avec, psi, [phi])) phis

fun solve cs =
    case mgu (Lst.flatmapmix explode cs) [] of
         Either.RIGHT sigma => Either.RIGHT [(sigma, Unit)]
       | Either.LEFT phi    => Either.LEFT  [phi]
```

```
type constraint = (name list * assertion * condition list) list


type solution =
  (condition list list, ((name * term) list * assertion) list)
   either

val solve : constraint -> solution
```

Constraint Solver

```
fun mgu [] sigma = Either.RIGHT sigma
    | mgu ((avec, Unit,  [True] )::cs) sigma =
      mgu cs sigma
    | mgu ((avec, Unit, [Eq (NextFreq a, NextFreq b)])::cs)
          sigma =
        mgu ((avec, Unit, [Eq (a,b)])::cs) sigma
    | mgu ((avec, Unit, [Eq (NextFreq a, Name b)])::cs)
          sigma =
        mgu ((avec, Unit, [Eq (Name b, NextFreq a)])::cs)
            sigma
    | mgu ((avec, Unit, [Eq (Name a, n)])::cs) sigma =
       if Name a = n then mgu cs sigma
         else
           if L.fresh a avec andalso
              freshL avec (supportT n) andalso
              L.fresh a (supportT n)
             then
               mgu (Constraint.subst [(a, n)] cs)
                   (composeSubst sigma (a, n))
```

Print ...

```
fun printN n = n

fun printT (Name n) = n
  | printT (NextFreq t) = "nextFreq(" ^ printT t ^ ")"

fun printC True = "T"
  | printC (Eq (t1, t2)) =
      (printT t1) ^ " = " ^ (printT t2)
fun printA _ = "1"
```

... Parse

```
    fun term () =
          (stok "nextFreq" >> stok "(" >>
          (delayed term) >>=
          (fn t => stok ")" >> return (NextFreq t)))
      </choice/>
          (Lex.identifier >>= return o Name)

 ")"
        fun name () = (Lex.identifier)

        fun cond () = (stok "T" >> return True)

        fun assr () = (stok "1" >> return Unit)
```

```
psi> sstep "nextFreq(x)"(f).P<f> | '"nextFreq(x)"<"nextFreq(y)">;
```

....

```
3 ---
  1 |>
    --|tau|-->

      Source:
        ("nextFreq(x)"(f). P<f>) |
          ('"nextFreq(x)"<"nextFreq(y)">)
      Constraint:
        {| "nextFreq(x) = nextFreq(x)" |}
      Solution:
        ([], 1)
      Derivative:
        (P<"nextFreq(y)">) | (0)
```

Left overs

- Symbolic Broadcast Semantics
- Bisimulation algorithm
- Sorts