Discover[i]

Component-based Parameterized Reasoning for Distributed Systems

Roopsha Samanta

PURDUE UNIVERSITY

@purdue_pl

Ben
Delaware

Suresh
Jagannathan

Milind
Kulkarni

Zhiyuan
Li

Samuel
Midkiff

Xiaokang
Qiu

Tiark
Rompf

Roopsha
Samanta

Xiangyu
Zhang

Research Agenda

Make it easier to write reliable programs using automated reasoning

Formal methods

Programming languages

Program repair

Program synthesis

Program verification

Concurrent systems

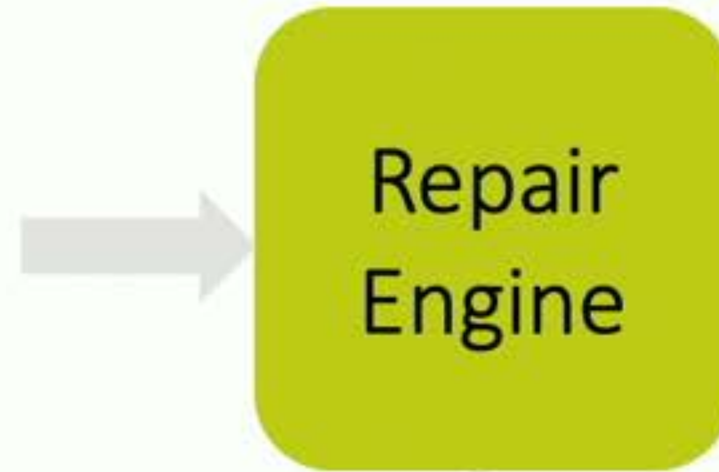
Distributed systems

Personalized education

Machine learning

Cost-Aware Program Repair

```
Power(int x, int i){
  if (i<=2)
     if (i=0)
        return 0
     elseif (i=1)
        return x
  else
     return pow(x,i)
}
```



$i>=0 \rightarrow Power(x,i)=x^i$

Joint work with Loris D'Antoni, Allen Emerson, Rishabh Singh and Oswaldo Olivo [CAV 2016, SAS 2014, FMCAD 2008]

Cost-Aware Program Repair

Syntactic + semantic program distances

```
Power(int x, int i){
  if (i<=2)
    if (i=0)
      return 0
    elseif (i=1)
      return x
  else
    return pow(x,i)
}
```

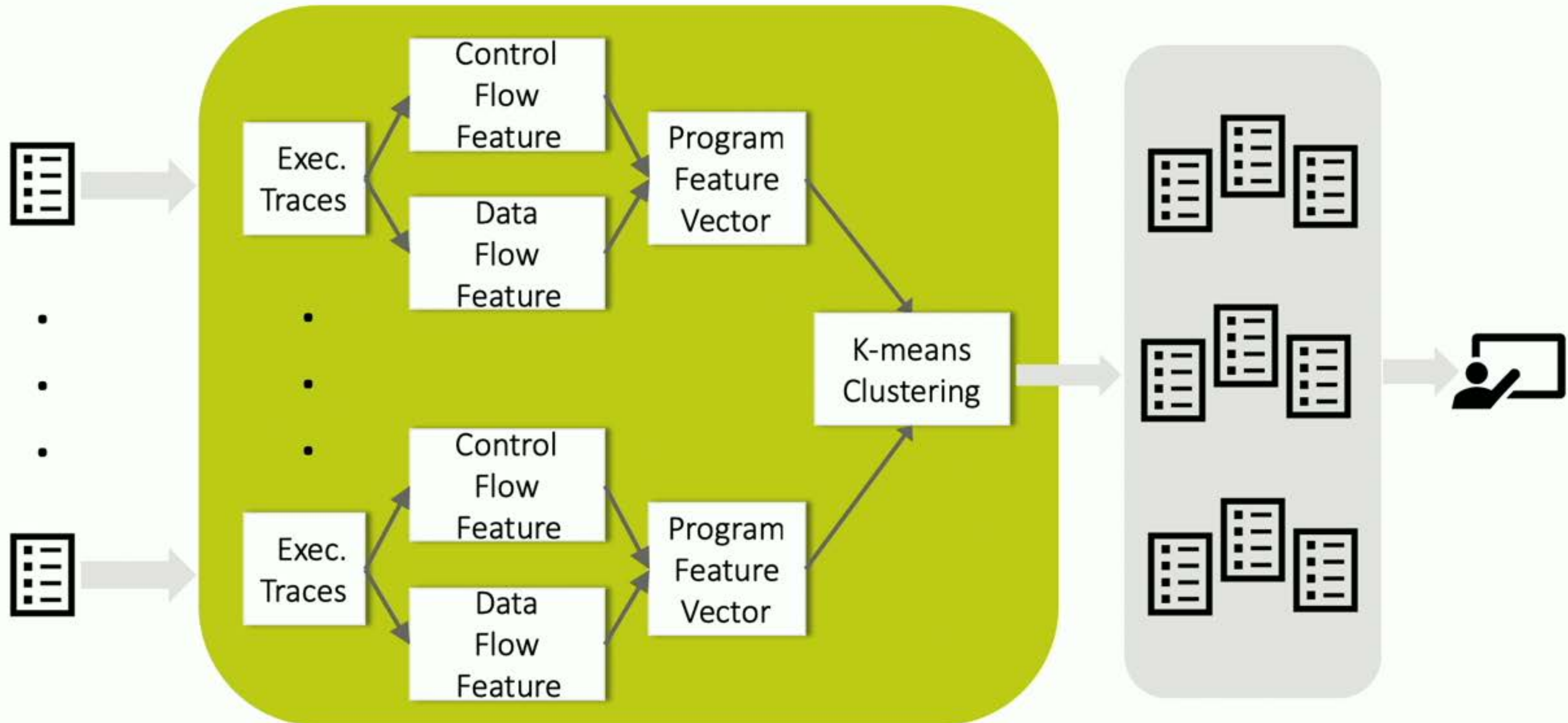
Repair Engine

```
Power(int x, int i){
  if (i<=2)
    if (i=0)
      return 1
    elseif (i=1)
      return x
  else
    return pow(x,i)
}
```

$i>=0 \rightarrow Power(x,i)=x^i$

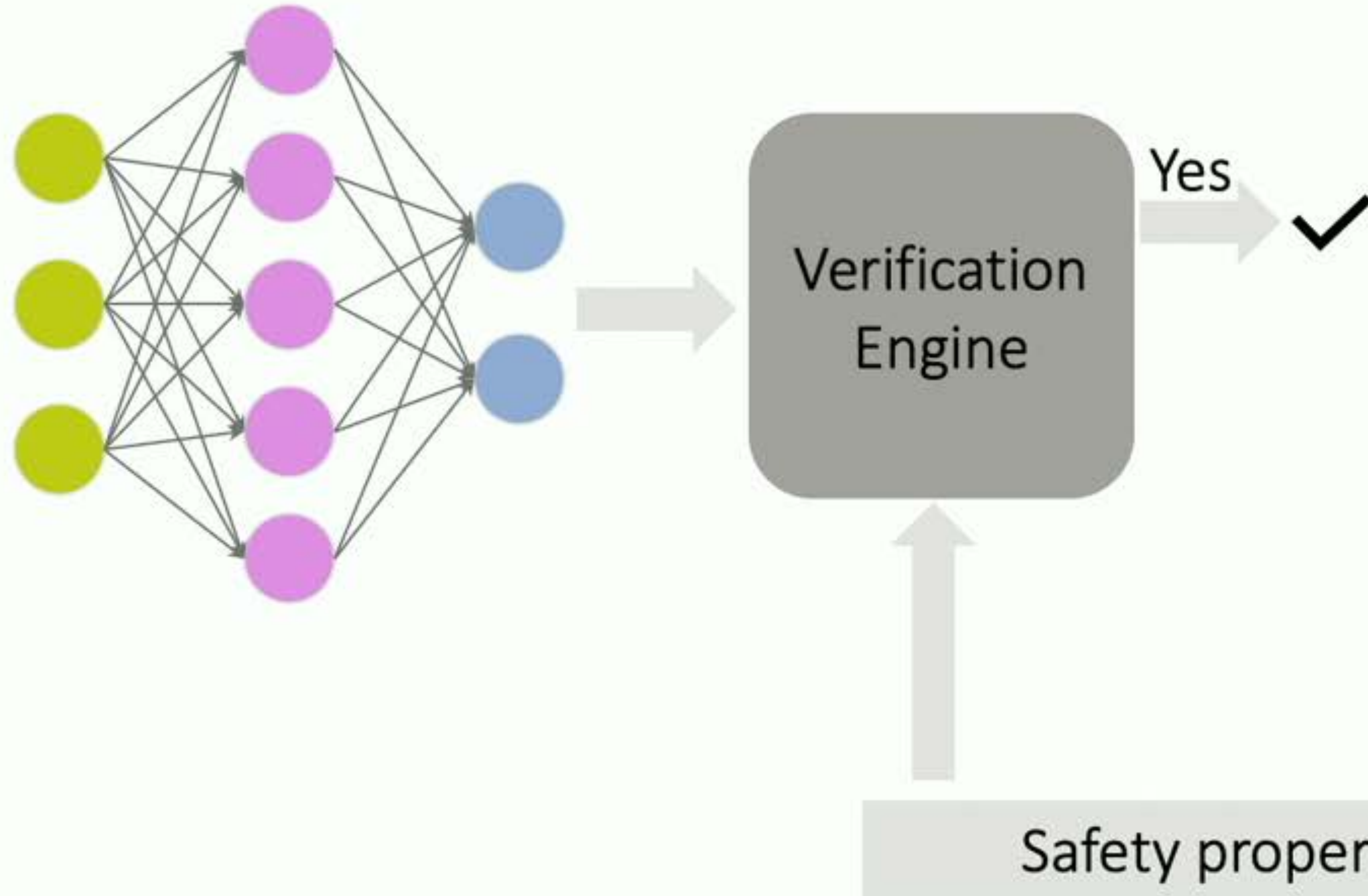
Joint work with Loris D'Antoni, Allen Emerson, Rishabh Singh and Oswaldo Olivo [CAV 2016, SAS 2014, FMCAD 2008]

Program Clustering with Quantitative Semantic Features



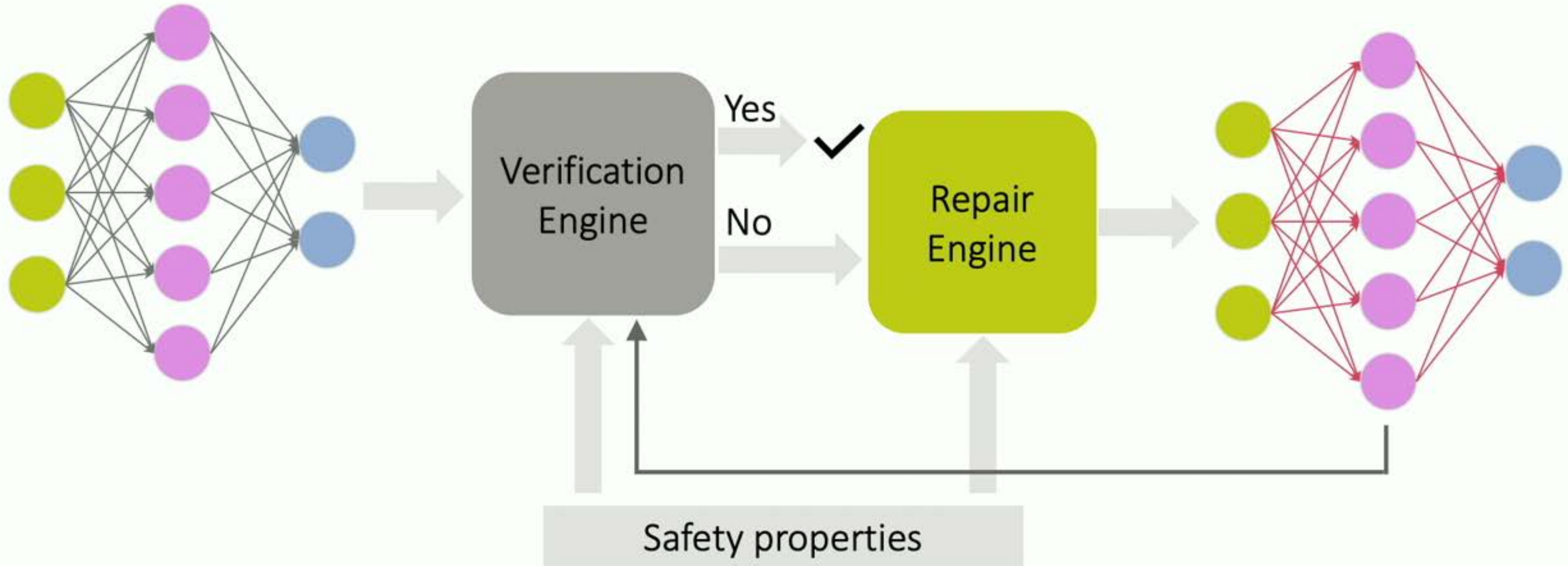
Joint work with David Perry, Dohyeong Kim and Xiangyu Zhang

Neural Network Repair



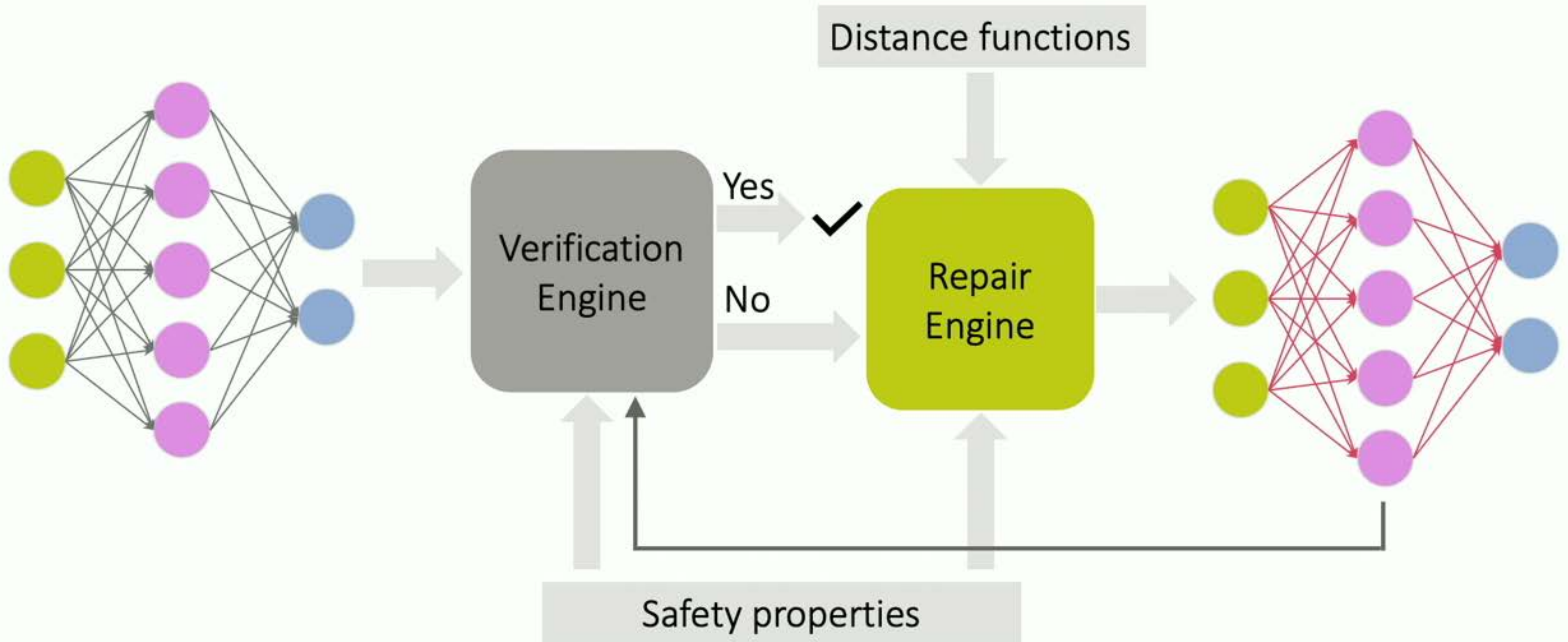
Joint work with Suresh Jagannathan and Xuankang Liu

Neural Network Repair



Joint work with Suresh Jagannathan and Xuankang Liu

Neural Network Repair



Joint work with Suresh Jagannathan and Xuankang Liu

Augmented example-based synthesis

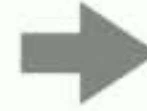
Partial program

```
int max(int x, int y, int z){
  int m = ??v;
  if (??e) m = ??v;
  if (??e) m = ??v;
  return m;
}
```

(a)

Example set E

```
( 0, 10,  2) ⇒ 10;
(-1, 10, 20) ⇒ 20;
(-1, -2, -3) ⇒ -1;
```



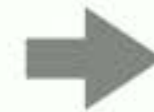
Synthesized program

```
int max(int x, int y, int z) {
  int m = x;
  if(y < z) m = z;
  if(m < y) m = y;
  return m;
}
```

(b)

Augmented example set

```
( 0, 10,  2) ⇒ 10; (-1, 10, 20) ⇒ 20; (-1, -2, -3) ⇒ -1;
( 0,  2, 10) ⇒ 10; (-1, 20, 10) ⇒ 20; (-1, -3, -2) ⇒ -1;
(10,  0,  2) ⇒ 10; (10, -1, 20) ⇒ 20; (-2, -1, -3) ⇒ -1;
(10,  2,  0) ⇒ 10; (10, 20, -1) ⇒ 20; (-2, -3, -1) ⇒ -1;
( 2,  0, 10) ⇒ 10; (20, -1, 10) ⇒ 20; (-3, -1, -2) ⇒ -1;
( 2, 10,  0) ⇒ 10; (20, 10, -1) ⇒ 20; (-3, -2, -1) ⇒ -1;
```



Synthesized program

```
int max(int x, int y, int z) {
  int m = z;
  if(z ≤ y) m = y;
  if(m  < x) m = x;
  return m;
}
```

(c)

Joint work with Shengwei An, Sasa Misailovic and Rishabh Singh

Synchronization Synthesis for Concurrent Programs

```
void open_dev()
  if (open==0)
     power_up();
  open := open+1;
  yield;

void close_dev()
  if (open>0)
     open := open-1;
     if (open==0)
        power_down();
  yield;
```

Synchronization Synthesizer

Preemption-safety

```
void open_dev()
 lock(l)
  if (open==0)
     power_up();
  open := open+1;
 unlock(l)
  yield;

void close_dev()
 lock(l)
  if (open>0)
     open := open-1;
     if (open==0)
        power_down();
 unlock(l)
  yield;
```

Joint work with Pavol Cerny, Ed Clarke, Ashutosh Gupta, Tom Henzinger, Arjun Radhakrishna, Leonid Ryzhyk and Thorsten Tarrach

[POPL 2015, CAV 2015, FMSD 2017]

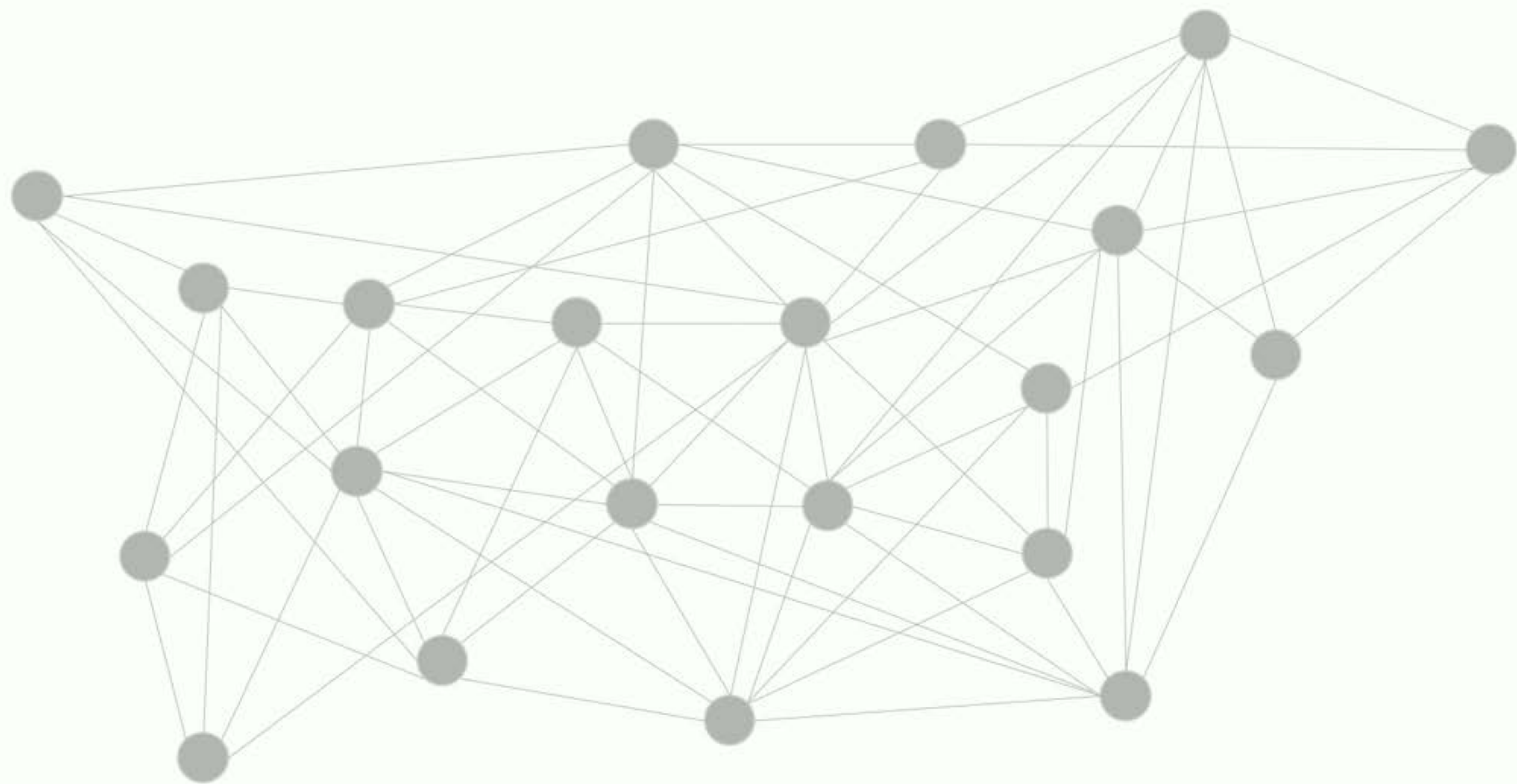
Discover[i]

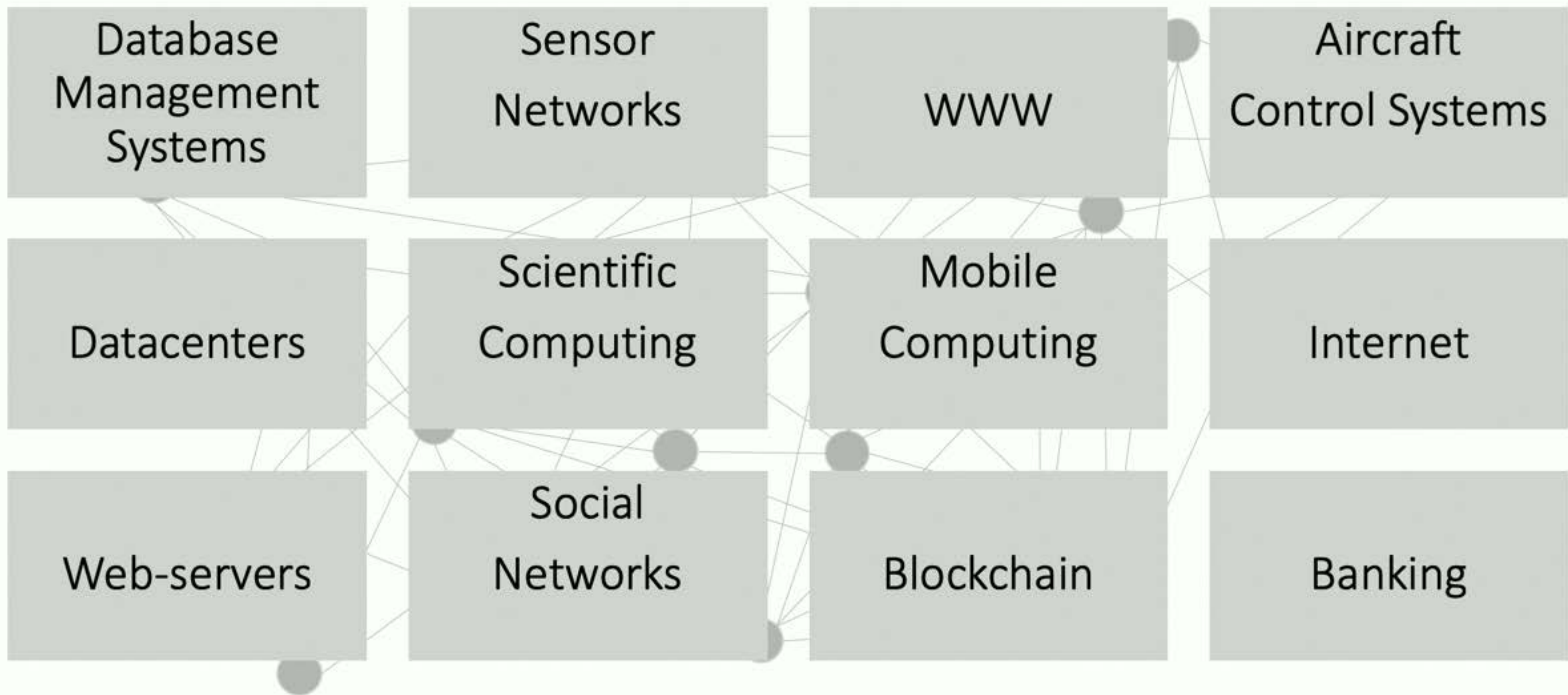
Nouraldin
Jaber

Swen
Jacobs

Milind
Kulkarni

Roopsha
Samanta





Database Management Systems
Sensor Networks
WWW
Aircraft Control Systems
Datacenters
Scientific Computing
Mobile Computing
Internet
Web-servers
Social Networks
Blockchain
Banking

What makes building correct distributed systems hard?

Complex building blocks

What makes building correct distributed systems hard?

Complex building blocks

Consensus

Atomic commit

Reliable broadcast

...

What makes building correct distributed systems hard?

Complex building blocks

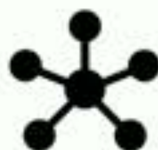
Consensus

[Sergey et al. 2018]
[Padon et al. 2016]
[Dragoi et al. 2016]
[Hawblitzel et al. 2015]
[Wilcox et al. 2015]
[Cousineau et al. 2012]
[Lamport 2002]

What makes building correct distributed systems hard?

Complex building blocks

Arbitrary number of processes

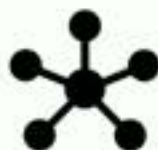


Parameterized verification is undecidable

What makes building correct distributed systems hard?

Complex building blocks

Arbitrary number of processes



Programmer errors

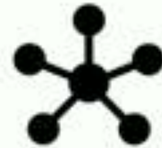
Too many moving parts!

Discover[i]

Challenges

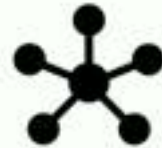
Complex building blocks

Arbitrary number of processes

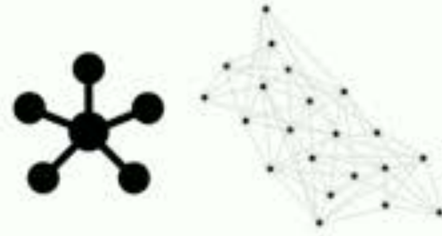


Programmer errors

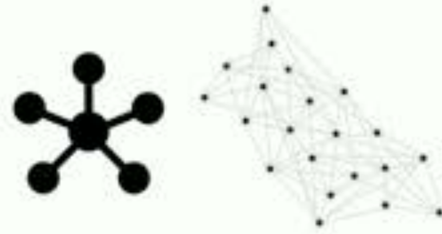
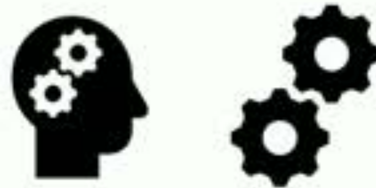
Discover[i]

Challenges		Solutions
Complex building blocks		Abstractions
Arbitrary number of processes		
Programmer errors		

Discover[i]

Challenges		Solutions
Complex building blocks		Abstractions
Arbitrary number of processes		Parameterized verification
Programmer errors		

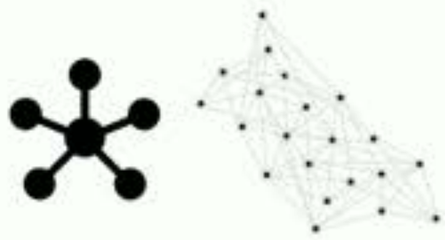
Discover[i]

Challenges		Solutions
Complex building blocks		Abstractions
Arbitrary number of processes		Parameterized verification
Programmer errors		Parameterized synthesis

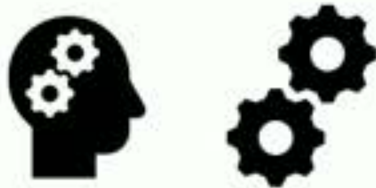
Discover[i]

Distributed protocols with consensus components

Abstractions



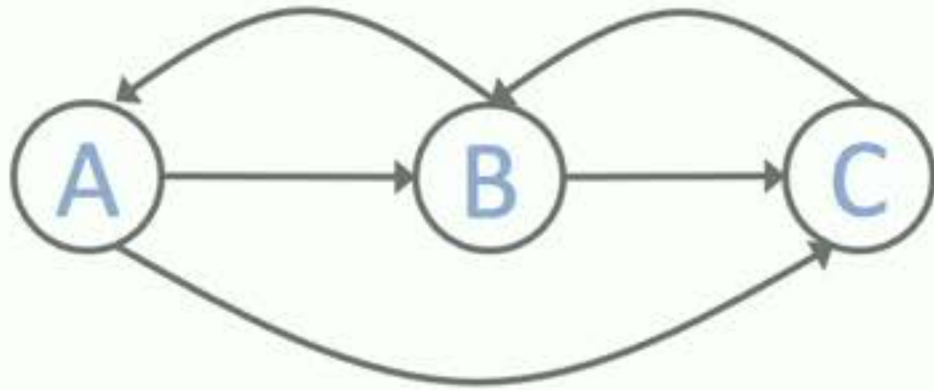
Parameterized verification



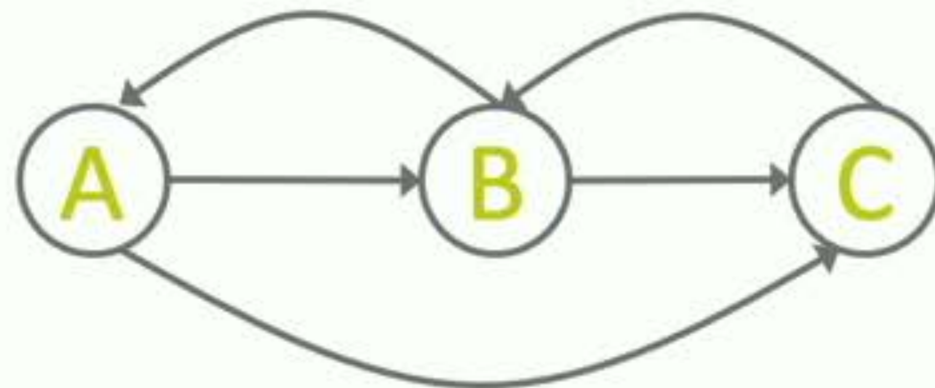
Parameterized synthesis

- **System model**: Distributed protocols
- **Properties**: LTL
- **Failures**: No process or channel failures
- **Communication**: Instantaneous

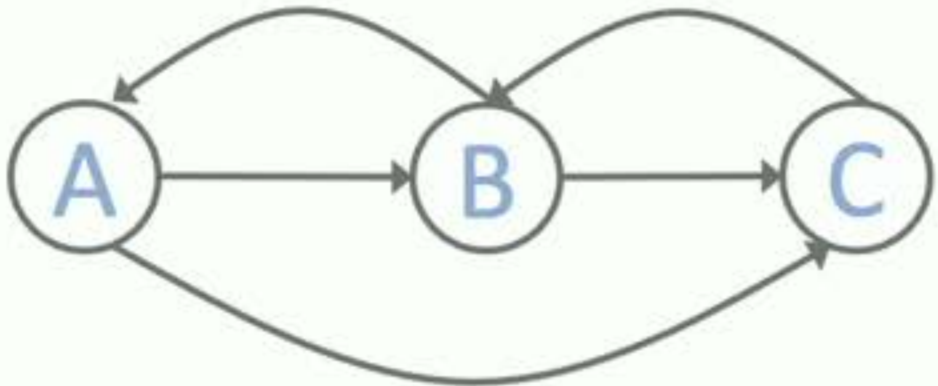
Basic protocols



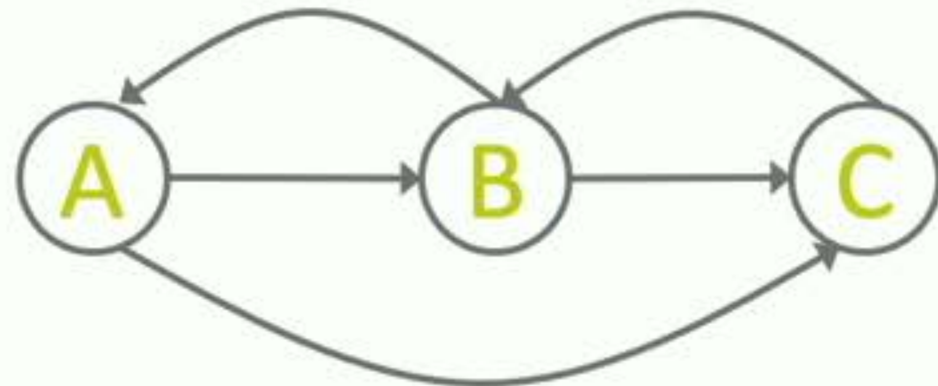
Identical FSMs



Basic protocols



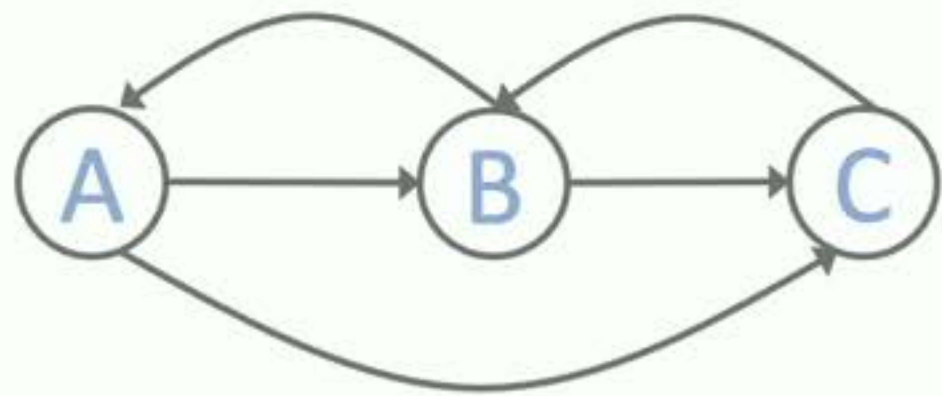
‖



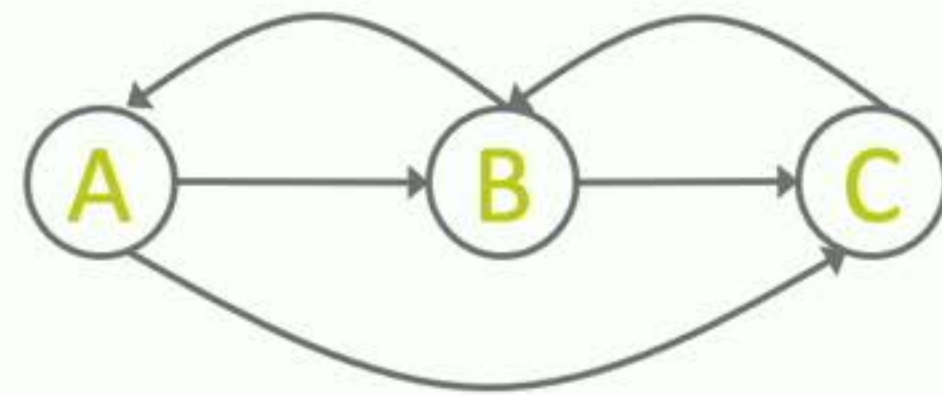
Identical FSMs

Interleaving semantics

Basic protocols

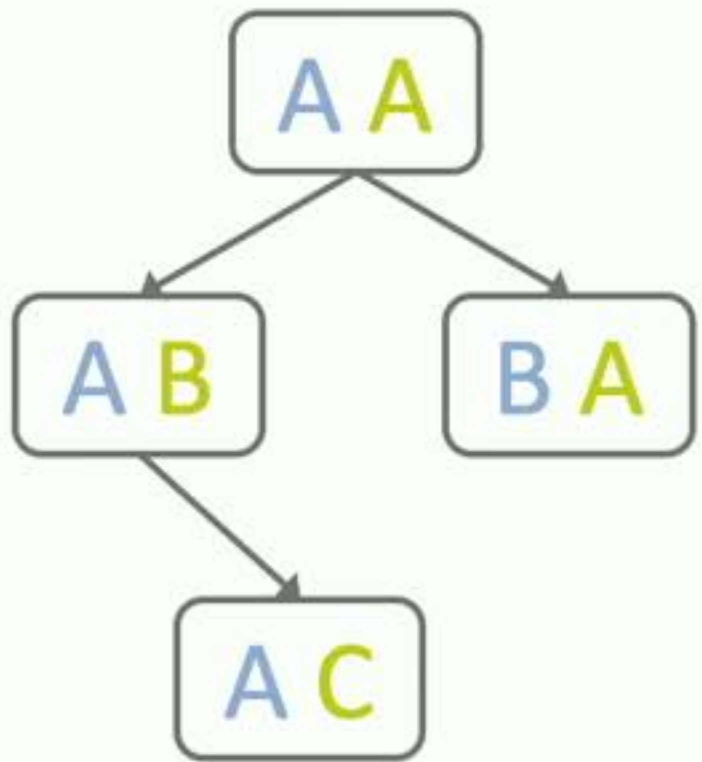


||

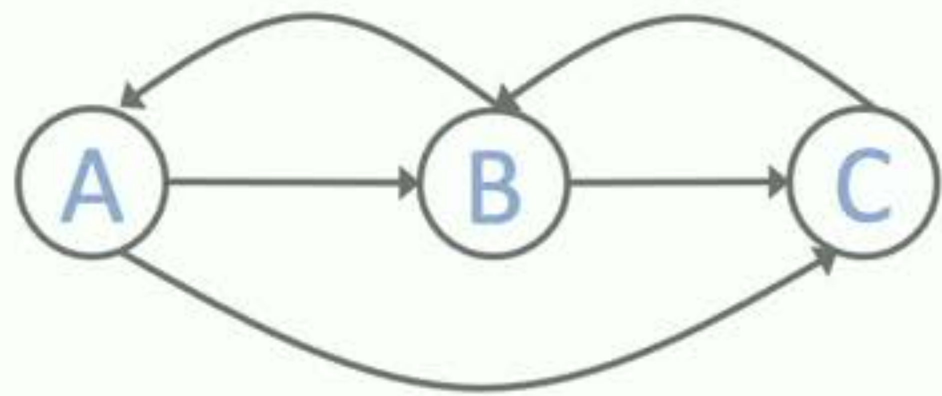


Identical FSMs

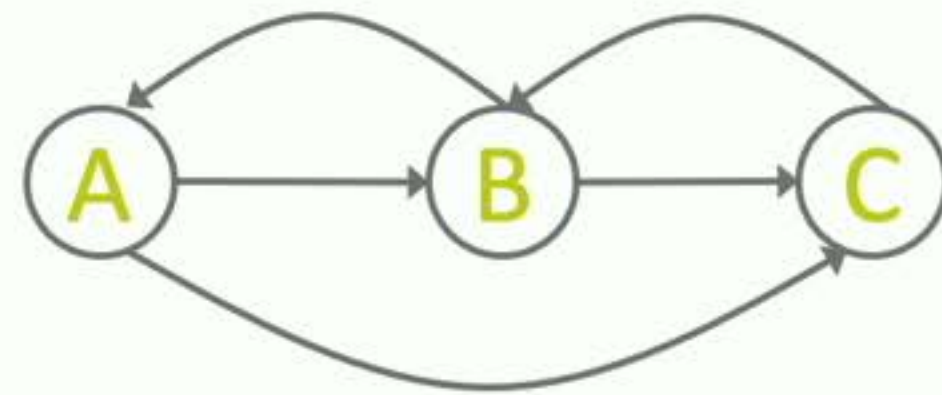
Interleaving semantics



Basic protocols



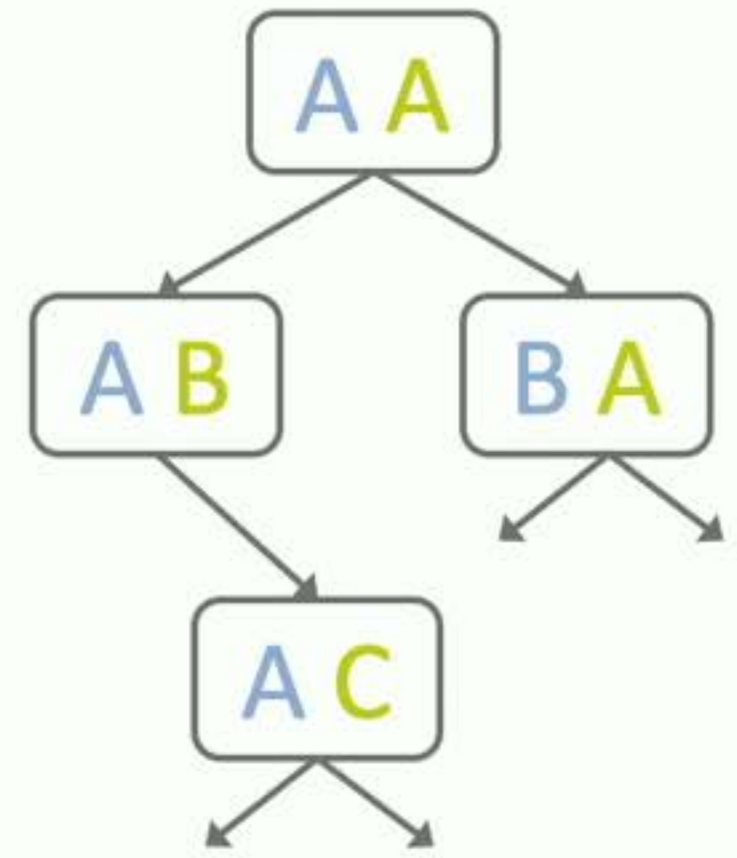
∥



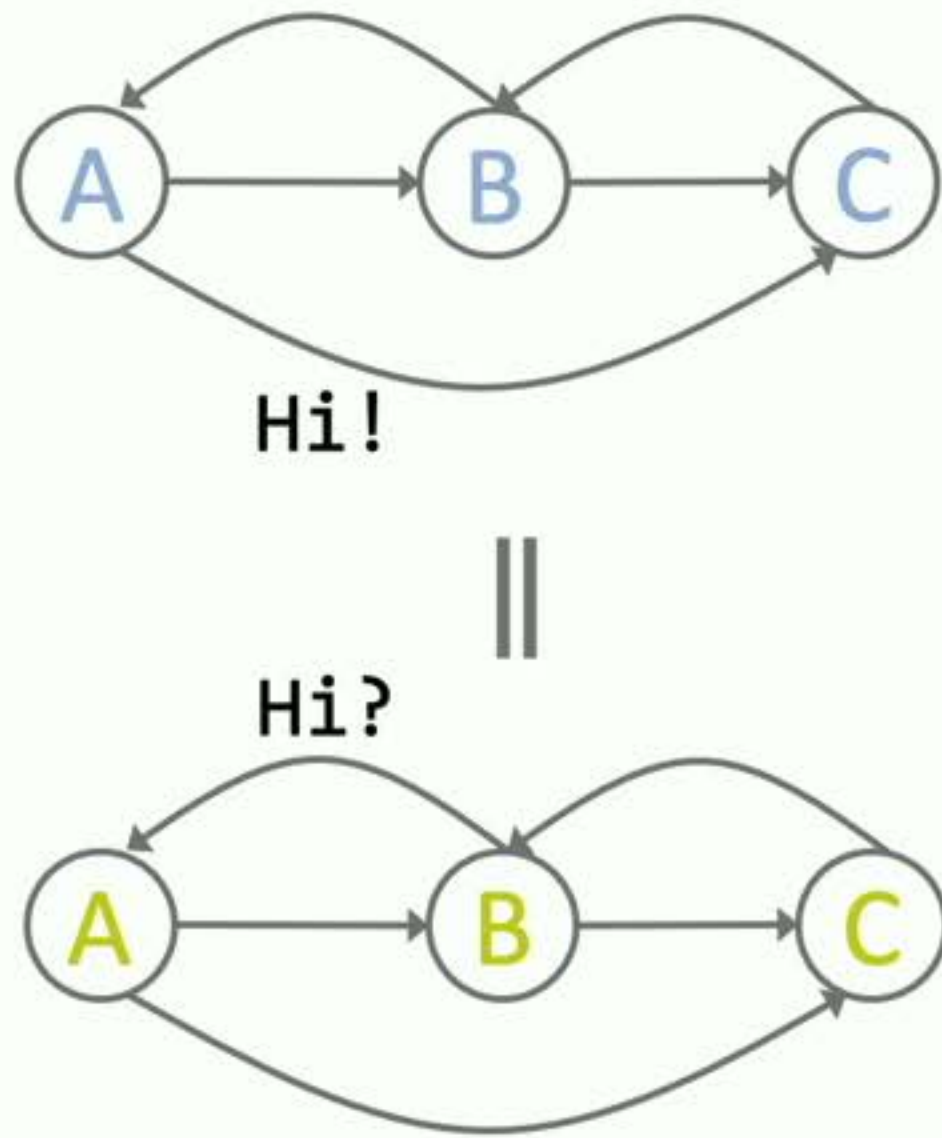
Identical FSMs

Interleaving semantics

Broadcasts
Peer-to-peer



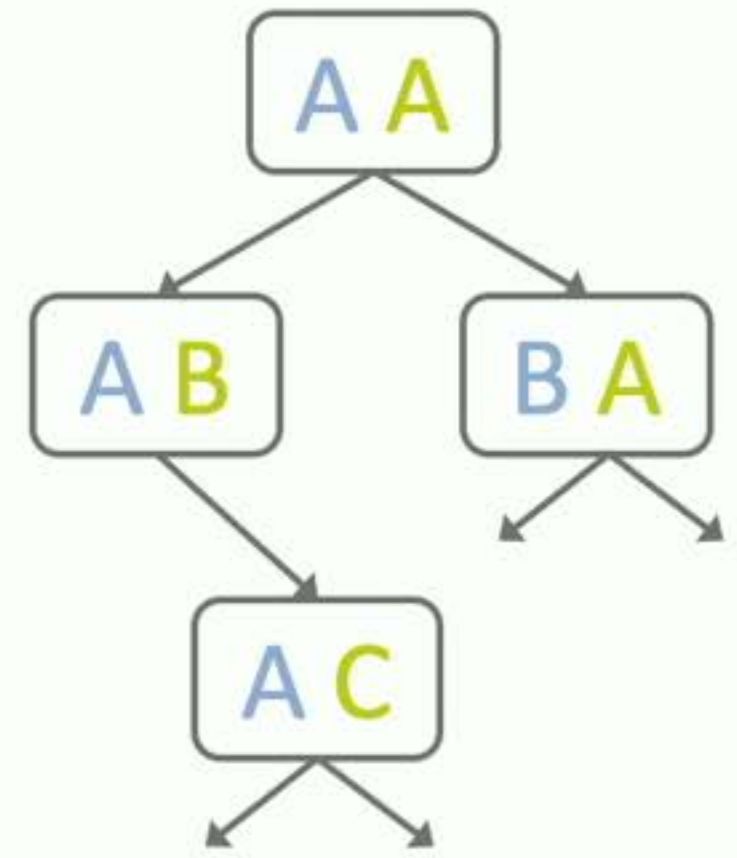
Basic protocols



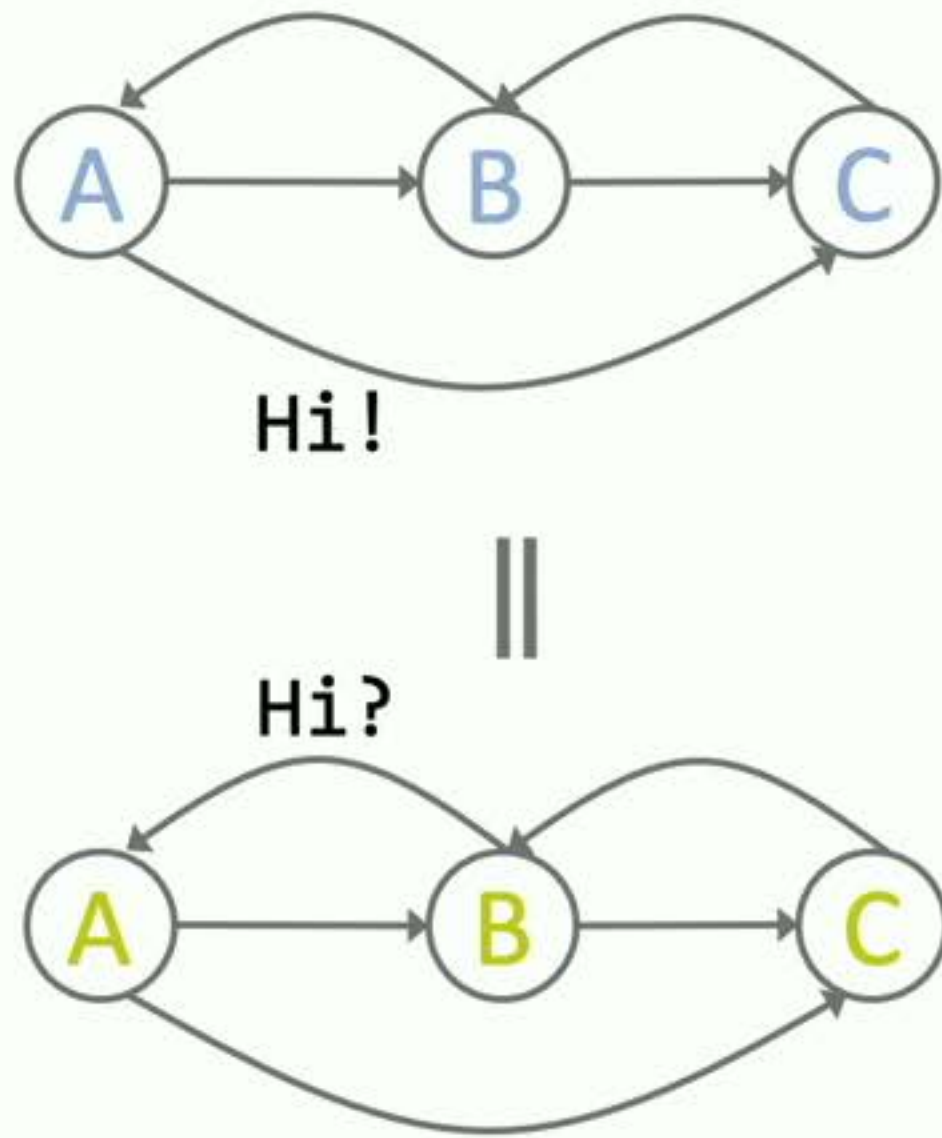
Identical FSMs

Interleaving semantics

Broadcasts
Peer-to-peer



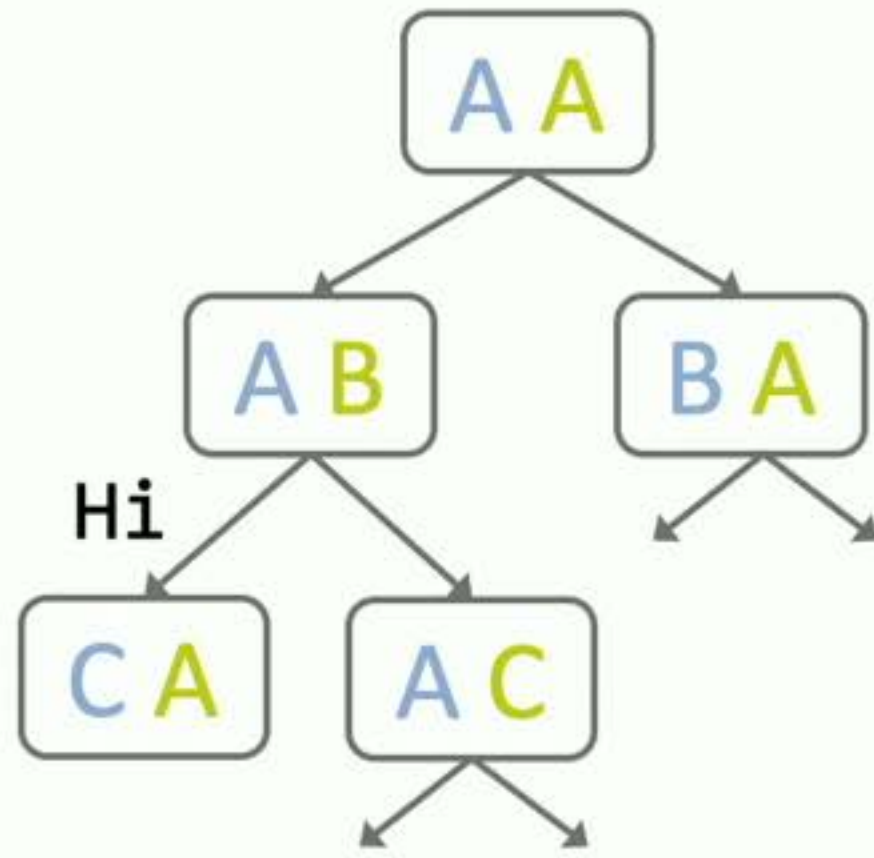
Basic protocols



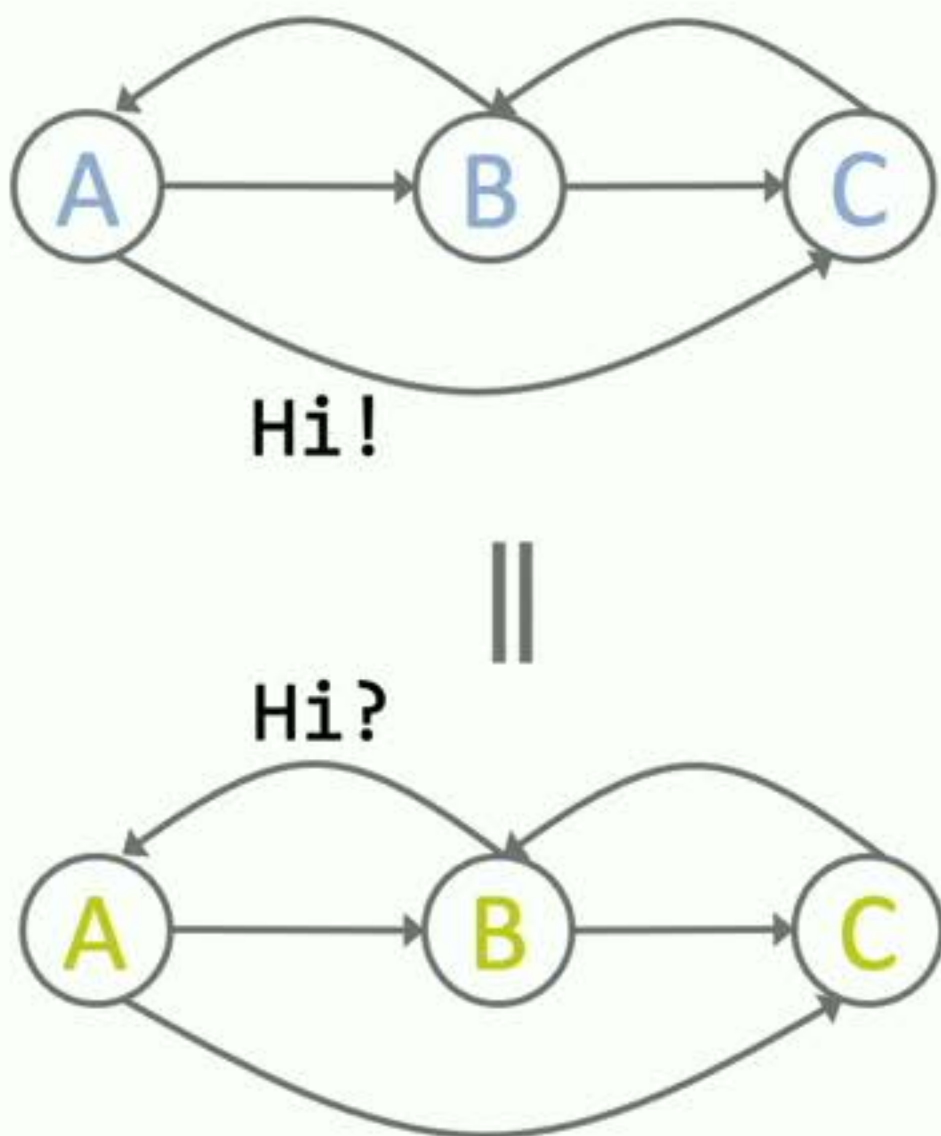
Identical FSMs

Interleaving semantics

Broadcasts
Peer-to-peer



Basic protocols

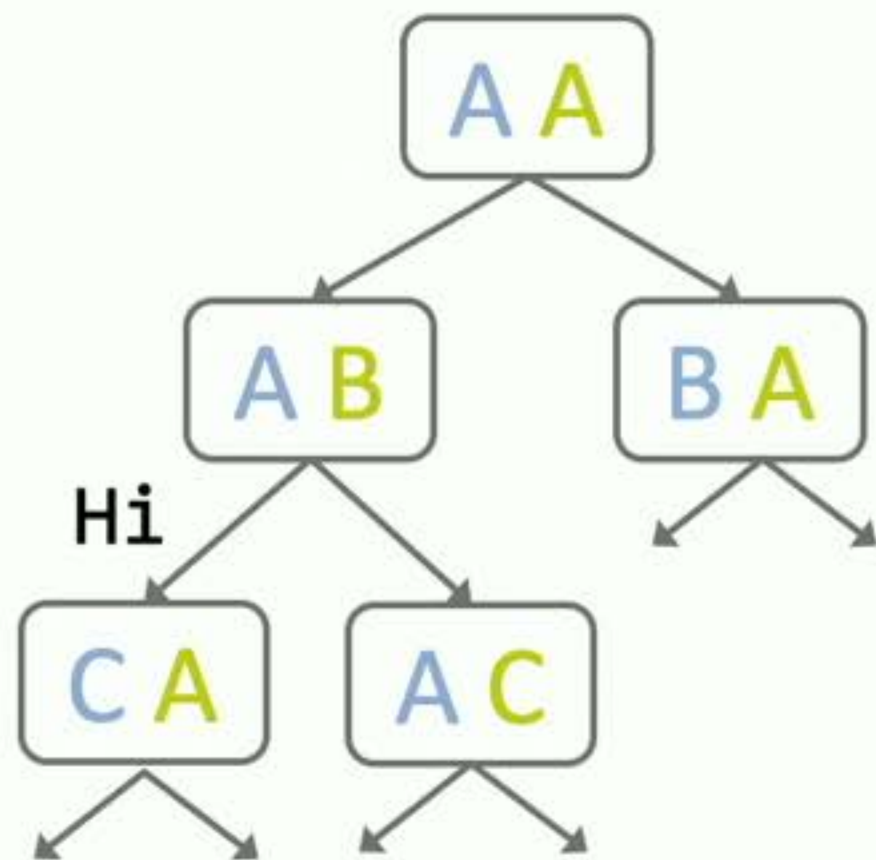


Identical FSMs

Interleaving semantics

Broadcasts
Peer-to-peer

Guarded commands



No process/channel failures
Instantaneous message delivery
LTL properties

Consensus protocol

cons(   )

Consensus protocol

Participants

```
cons({1,2,4}    )
```

Consensus protocol

Participants

```
cons({1,2,4},2)
```

Cardinality

Consensus protocol

Participants

```
cons({1,2,4},2)= {1,2}
```

Cardinality

Winners

Consensus protocol

Participants

$$\text{cons}(\{1,2,4\},2)=\{\{1,2\},\{1,4\},\{2,4\}\}$$

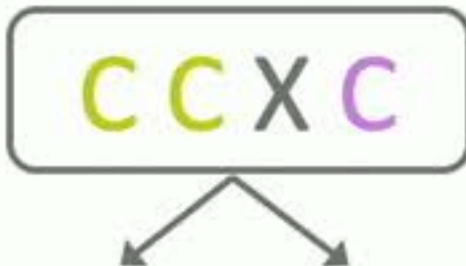
Cardinality

Winners

```
cons({1,2,4},2) = {{1,2},{1,4},{2,4}}
```

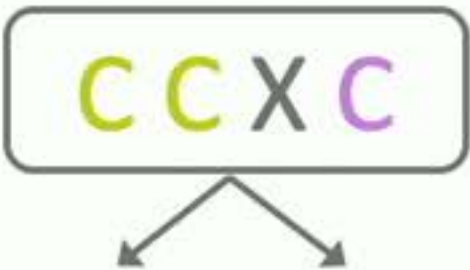
```
On ⟨Leader, L, n⟩:
  if n > n_L :
    leader := L , n_L := n
    if self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = ⟨⟩; las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N ; l_c := 0,
      send ⟨Prepare, n_L, l_d, n_a⟩ to all π – { self }
      acks[L] := (n_a ,suffix(v_a , l_d))
      lds[self] := l_d; n_prom := n_L
    else:
      state = (follower, state[2])

On ⟨Prepare, n_L , ld, n⟩ from L:
  if n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    suffx := if n_a ≥ n : suffix(v_a , ld) else ⟨⟩
    send ⟨Promise, n_L, n_a, suffx, l_d⟩ to L
```

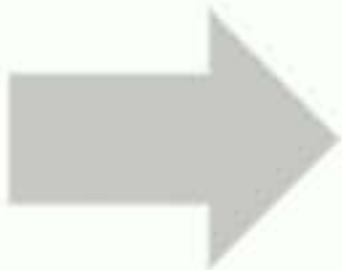
```
  acks[a] := (n_a, suffx_a) , lds[a] := ld
  P := {p in π : acks[p] ≠ ⊥}
  if |P| = ⌈(N+1)/2⌉:
    (k, suffx) := max({acks[p]: p in P}) // adopt v
    v_a = prefix(v_a, l_d) + suffx + propCmds;
    las[self] := |v_a|
    propCmds := ⟨⟩; state := (leader, accept)
    for p in π- {self} and lds[p] ≠ ⊥:
      sufx := suffix(v_a, lds[p])
      send ⟨AcceptSync, n_L, sufx, lds[p]⟩ to p

On ⟨Promise, n, n_a, suffx_a, ld⟩ from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send ⟨AcceptSync, n_L, suffix(v_a, lds[a]), lds[a]⟩ to a
  if l_c ≠ 0:
    send ⟨Decide, l_d, n_L⟩ to a
```



cons({1,2,4},2) = {{1,2},{1,4},{2,4}}

```
On ⟨Leader, L, n⟩:
  if n > n_L :
    leader := L ; n_L := n
    if self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = ⟨⟩; las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N; l_c := 0;
      send ⟨Prepare, n_L, l_d, n_a⟩ to all π − { self }
      acks[L] := (n_a, suffix(v_a, l_d))
      lds[self] := l_d; n_prom := n_L
    else:
      state = (follower, state[2])

On ⟨Prepare, n_L, ld, n⟩ from L:
  if n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    suffix := if n_a ≥ n : suffix(v_a, ld) else ⟨⟩
    send ⟨Promise, n_L, n_a, suffix, l_a⟩ to L

    acks[a] := (n_a, suffix_a) , lds[a] := ld
    P := {p in π : acks[p] ≠ ⊥}
    if |P| = ⌈(N+1)/2⌉:
      (k, suffix) := max({acks[p]: p in P}) // adopt v
      v_a = prefix(v_a, l_d) + suffix + propCmds;
      las[self] := |v_a|
      propCmds := ⟨⟩; state := (leader, accept)
      for p in π- {self} and lds[p] ≠ ⊥:
        sufx := suffix(v_a, lds[p])
        send ⟨AcceptSync, n_L, sufx, lds[p]⟩ to p

On ⟨Promise, n, n_a, suffix_a, ld⟩ from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send ⟨AcceptSync, n_L, suffix(v_a, lds[a]), lds[a]⟩ to a
  if l_c ≠ 0:
    send ⟨Decide, l_c, n_L⟩ to a
```





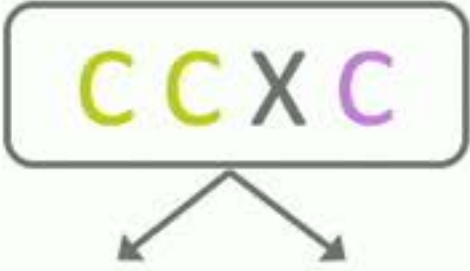
Primitive

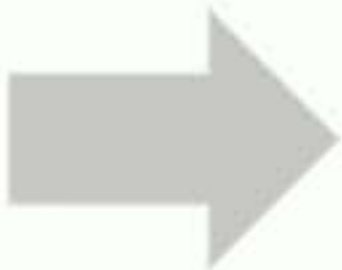
+

Specification

```
cons({1,2,4},2) = {{1,2},{1,4},{2,4}}
```

```
On ⟨Leader, L, n⟩:
  If  n > n_L :
    leader := L ; n_L := n
    If  self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = ⟨⟩; las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N; l_c := 0;
      send ⟨Prepare, n_L, l_d, n_a⟩ to all π – { self }
      acks[L] := (n_a ,suffix(v_a , l_d))
      lds[self] := l_d; n_prom := n_L
  else:
    state = (follower, state[2])

On ⟨Prepare, n_L , ld, n⟩ from L:
  If n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    suffix := If n_a ≥ n : suffix(v_a , ld) else ⟨⟩
    send ⟨Promise, n_L, n_a, suffix, l_d⟩ to L

  acks[a] := (n_a, suffix_a) , lds[a] := ld
  P := {p in π : acks[p] ≠ ⊥}
  If |P| = ⌈(N+1)/2⌉:
    (k, suffix) := max({acks[p]: p in P}) // adopt v
    v_a = prefix(v_a, l_d) + suffix + propCmds;
    las[self] := |v_a|
    propCmds := ⟨⟩; state := (leader, accept)
    for p in π- {self} and lds[p] ≠ ⊥:
      sufx := suffix(v_a, lds[p])
      send ⟨AcceptSync, n_L, sufx, lds[p]⟩ to p

On ⟨Promise, n, n_a, suffix_a, ld⟩ from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send ⟨AcceptSync, n_L , suffix(v_a, lds[a]), lds[a]⟩ to a
  if l_c ≠ 0:
    send ⟨Decide, l_c, n_L⟩ to a
```





choose

Primitive

+

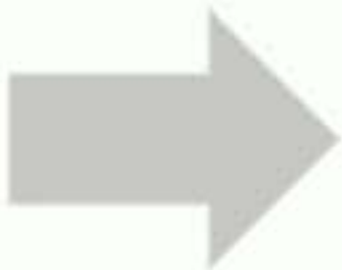
Specification

- Consistent Participants
- Consistent Winners

`cons({1,2,4},2) = {{1,2},{1,4},{2,4}}`

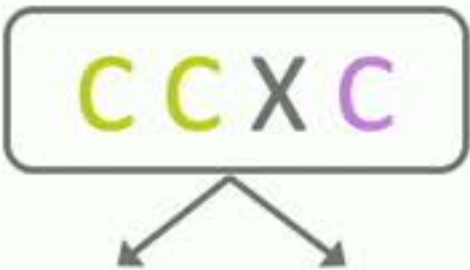
```
On ⟨Leader, L, n⟩:
  if n > n_L :
    leader := L , n_L := n
    if self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = ⟨⟩; las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N; l_c := 0;
      send ⟨Prepare, n_L, l_d, n_a⟩ to all π – { self }
      acks[L] := (n_a, suffix(v_a, l_d))
      lds[self] := l_d; n_prom := n_L
  else:
    state = (follower, state[2])

On ⟨Prepare, n_L , ld, n⟩ from L:
  if n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    suffix := if n_a ≥ n : suffix(v_a , ld) else ⟨⟩
    send ⟨Promise, n_L, n_a, suffix, l_a⟩ to L
```

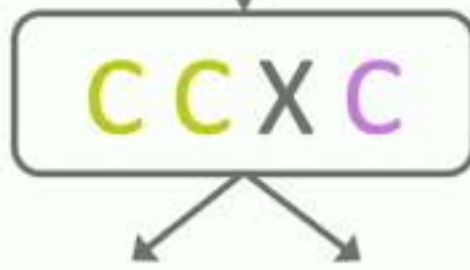
```
  acks[a] := (n_a, suffx_a) , lds[a] := ld
  P := {p in π : acks[p] ≠ ⊥}
  if |P| = ⌈(N+1)/2⌉:
    (k, suffx) := max({acks[p]: p in P}) // adopt v
    v_a = prefix(v_a, l_d) + suffx + propCmds;
    las[self] := |v_a|
    propCmds := ⟨⟩; state := (leader, accept)
    for p in π- {self} and lds[p] ≠ ⊥:
      sufx := suffix(v_a, lds[p])
      send ⟨AcceptSync, n_L, sufx, lds[p]⟩ to p

On ⟨Promise, n, n_a, suffx_a, ld⟩ from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send ⟨AcceptSync, n_L , suffix(v_a, lds[a]), lds[a]⟩ to a
  if l_c ≠ 0:
    send ⟨Decide, l_c, n_L⟩ to a
```



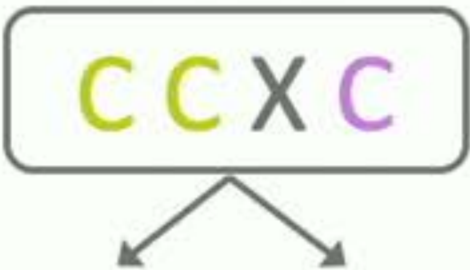
choose

Atomic






```
cons({1,2,4},2) = {{1,2},{1,4},{2,4}}
```



C C X C

choose

choose

choose

Non-deterministic

Atomic

C C X C

C C X C

C C X C

cons({1,2,4},2) = {{1,2},{1,4},{2,4}}

C C X C

Consistent Participants

C C X C

```
On (Leader, L, n):
  if n > n_L :
    leader := L , n_L := n
    if self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = (); las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N; l_c := 0
      send (Prepare, n_L, l_d, n_a) to all π − { self }
      acks[L] := (n_a, suffix(v_a, l_d))
      lds[self] := l_d; n_prom := n_L
    else:
      state = (follower, state[2])

On (Prepare, n_L, ld, n) from L:
  if n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    sufx := if n_a ≥ n : suffix(v_a, ld) else ()
    send (Promise, n_L, n_a, sufx, l_d) to L

  acks[a] := (n_a, suffix_a) , lds[a] := ld
  P := {p in π : acks[p] ≠ ⊥}
  if |P| = ⌈(N+1)/2⌉:
    (k, suffix) := max({acks[p]: p in P}) // adopt v
    v_a := prefix(v_a, l_d) + suffix + propCmds;
    las[self] := |v_a|
    propCmds := (); state := (leader, accept)
    for p in π- {self} and lds[p] ≠ ⊥ :
      sufx := suffix(v_a, lds[p])
      send (AcceptSync, n_L, sufx, lds[p]) to p

On (Promise, n, n_a, suffix_a, ld) from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send (AcceptSync, n_L, suffix(v_a, lds[a]), lds[a]) to a
  if l_c ≠ 0:
    send (Decide, l_d, n_L) to a
```

C C X C

C C X C

choose

choose

choose

C C X C

C C X C

C C X C

cons({1,2,4},2) = {{1,2},{1,4},{2,4}}

Consistent Participants

C C X C

X C X C

C C X C

```
On ⟨Leader, L, n⟩:
  if n > n_L :
    leader := L , n_L := n
    if self = L and n_L > n_prom :
      state := (leader, prepare)
      propCmds = ⟨⟩; las := [0]^N; lds := [⊥]^N
      acks := [⊥]^N ; l_c := 0;
      send ⟨Prepare, n_L , l_d, n_a⟩ to all π – { self }
      acks[L] := (n_a ,suffix(v_a , l_d))
      lds[self] := l_d; n_prom := n_L
    else:
      state = (follower, state[2])

On ⟨Prepare, n_L , ld, n⟩ from L:
  If n_prom < n_L:
    n_prom := n_L ; state := (follower, prepare)
    suffix := if n_a ≥ n : suffix(v_a , ld) else ⟨⟩
    send ⟨Promise, n_L , n_a, suffix, l_a ⟩ to L

    acks[a] := (n_a, suffix_a) , lds[a] := ld
    P := {p in π : acks[p] ≠ ⊥}
    If |P| = ⌈(N+1)/2⌉:
      (k, suffix) := max({acks[p]: p in P}) // adopt v
      v_a = prefix(v_a, l_d) + suffix + propCmds;
      las[self] := |v_a|
      propCmds := ⟨⟩; state := (leader, accept)
      for p in π- {self} and lds[p] ≠ ⊥:
        sufx := suffix(v_a, lds[p])
        send ⟨AcceptSync, n_L , sufx, lds[p]⟩ to p

On ⟨Promise, n, n_a, suffix_a, ld⟩ from a
  s.t. n = n_L and state = (leader, accept):
  lds[a] := ld
  send ⟨AcceptSync, n_L , suffix(v_a, lds[a]), lds[a]⟩ to a
  if l_c ≠ 0:
    send ⟨Decide, l_c, n_L ⟩ to a
```

C C X C

choose

choose

choose

C C X C

C C X C

C C X C

C C X C

cons({1,2,4},2) = {{1,2},{1,4},{2,4}}

X C X C

C C X C

Consistent Participants

C C X C

choose

choose

choose

C C X C

Consistent Winners

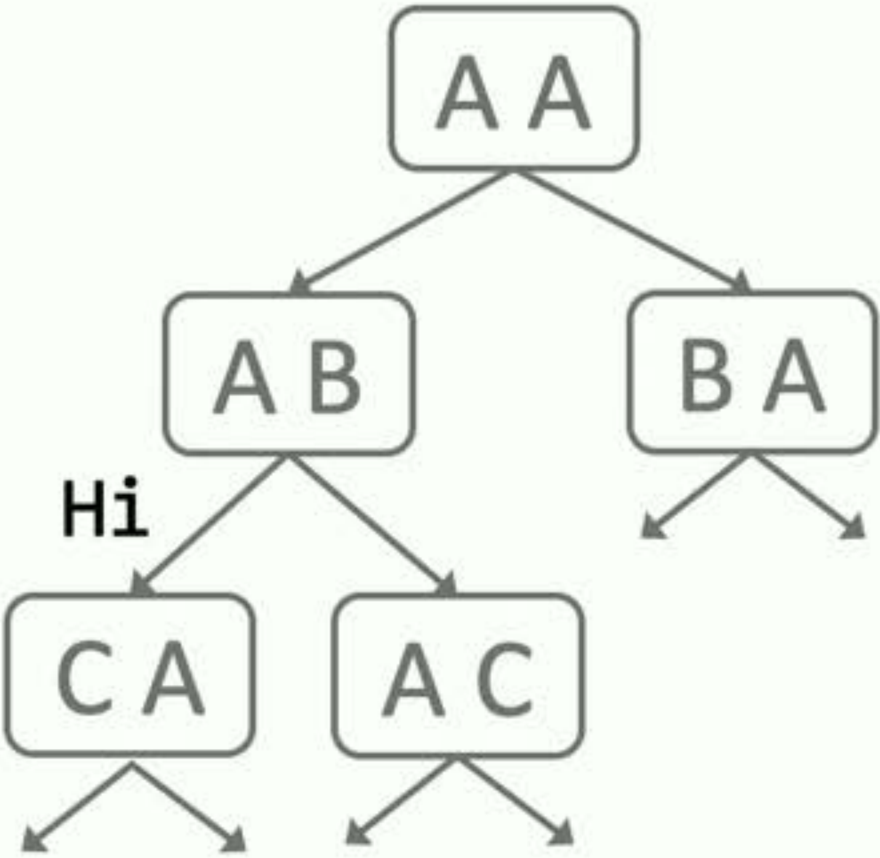
C C X C

C C X C

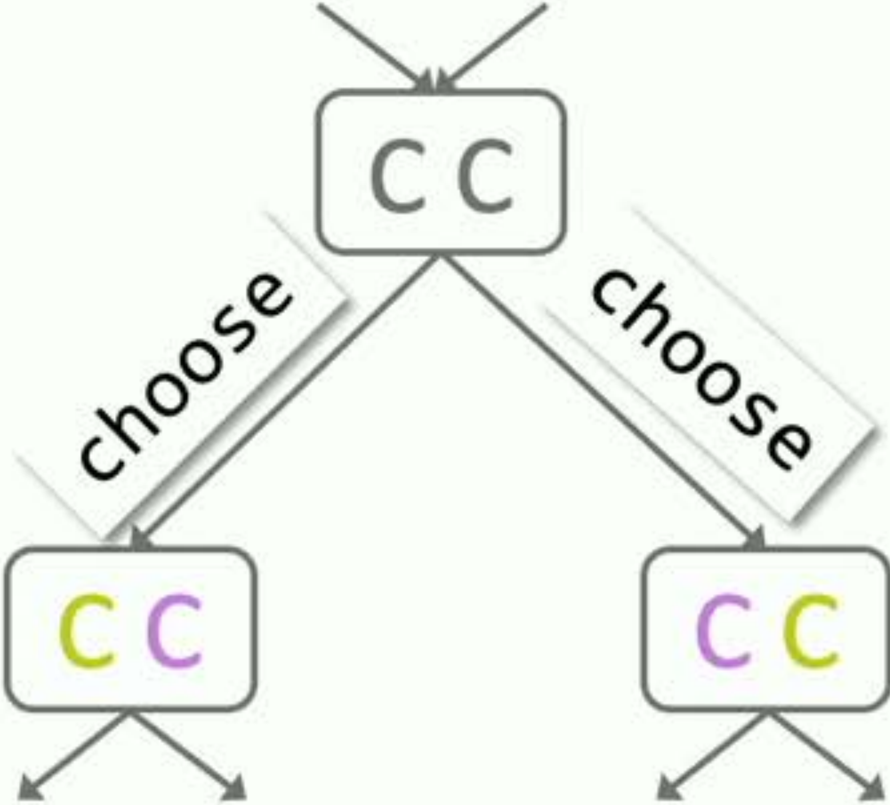
C C X C

$$\texttt{cons(\{1,2,4\},2) = \{\{1,2\},\{1,4\},\{2,4\}\}}$$

choose protocols



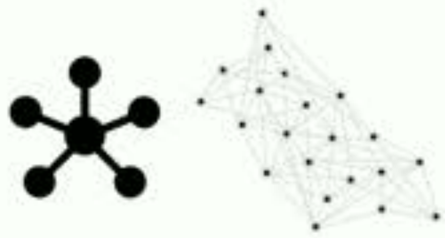
+



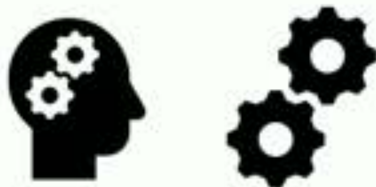
Discover[i]

Distributed protocols with consensus components

Abstraction

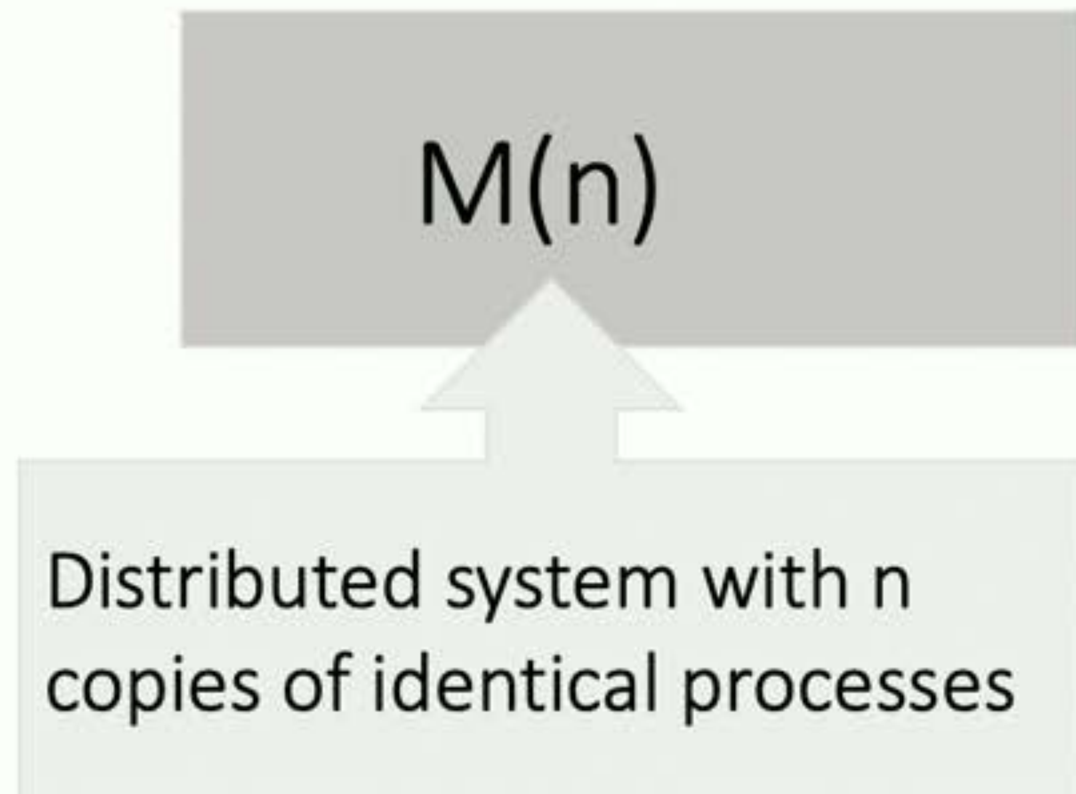


Parameterized verification



Parameterized synthesis

Parameterized Model Checking Problem (PMCP)



Parameterized Model Checking Problem (PMCP)

$\forall n.\ M(n) \models \phi\ ?$

Parameterized Model Checking Problem (PMCP)

Parameterized system

$\forall n.\ M(n) \models \phi$?

Undecidable

Parameterized Model Checking Problem (PMCP)

Decidable fragments

Communication primitives

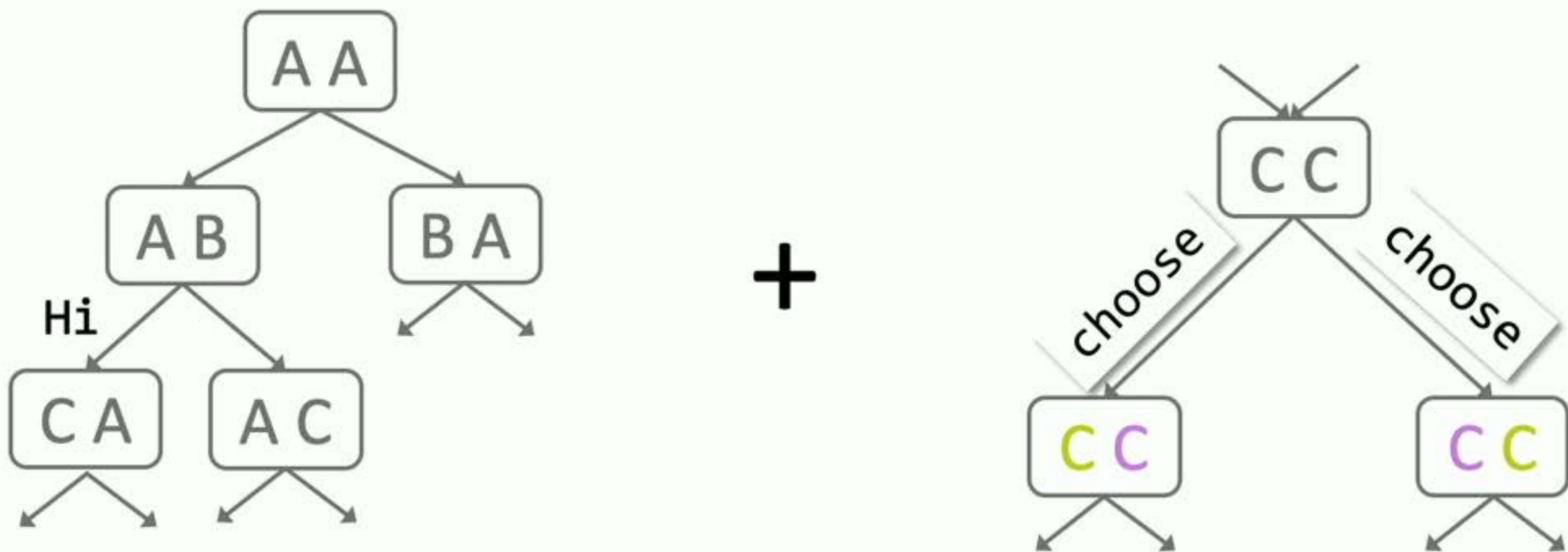
Network topology

Specification

Parameterized Model Checking Problem (PMCP)

Decidable fragments	Broadcast protocols	Guarded protocols
Communication primitives	Broadcasts	Global guards
Network topology	Clique	Clique
Specification	Safety	Safety + Liveness
	[Esparza et al. 1999]	[EmersonKahlon 2000]

choose protocols



Not subsumed by existing decidability results

Key result:

PMCP is decidable for the **choose** model w.r.t. safety properties.

Decidability of PMCP for Guarded Broadcast (GB) protocols

"partially synchronous round-based protocols"

Simulation equivalence b/w GB protocols and choose protocols

Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:

$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$

Cut-offs c for PMCP for GB protocols

$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$

Decidability of PMCP for Guarded Broadcast (GB) protocols
"partially synchronous round-based protocols"

Reduction to well-structured transition systems

Simulation equivalence b/w GB protocols and choose protocols
Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:
$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$

Predicate and counter abstraction

Cut-offs c for PMCP for GB protocols
$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$

Analysis of model checking algorithm

$M_l(c) \vDash \phi$

Decidability of PMCP for Guarded Broadcast (GB) protocols

"partially synchronous round-based protocols"

Simulation equivalence b/w GB protocols and choose protocols

Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:

$$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$$

Cut-offs c for PMCP for GB protocols

$$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$$

Decidability of PMCP for Guarded Broadcast (GB) protocols

"partially synchronous round-based protocols"

Simulation equivalence b/w GB protocols and choose protocols

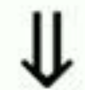
Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:

$$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$$

Cut-offs c for PMCP for GB protocols

$$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$$

$M_l(c) \vDash \phi$



$M_h(c) \vDash \phi$

Decidability of PMCP for Guarded Broadcast (GB) protocols

"partially synchronous round-based protocols"

Simulation equivalence b/w GB protocols and choose protocols

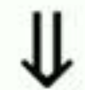
Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:

$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$

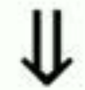
Cut-offs c for PMCP for GB protocols

$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$

$M_l(c) \vDash \phi$



$M_h(c) \vDash \phi$



$\forall n.\ M_h(n) \vDash \phi$

Decidability of PMCP for Guarded Broadcast (GB) protocols

"partially synchronous round-based protocols"

Simulation equivalence b/w GB protocols and choose protocols

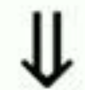
Given **choose** protocol M_l, can construct a GB protocol $M_h \simeq M_l$:

$\forall n.\ M_h(n) \vDash \phi \leftrightarrow M_l(n) \vDash \phi$

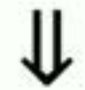
Cut-offs c for PMCP for GB protocols

$M_h(c) \vDash \phi \leftrightarrow \forall n.\ M_h(n) \vDash \phi$

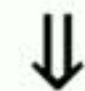
$M_l(c) \vDash \phi$



$M_h(c) \vDash \phi$



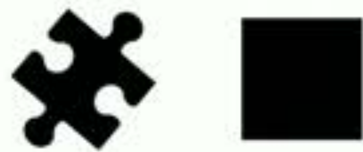
$\forall n.\ M_h(n) \vDash \phi$



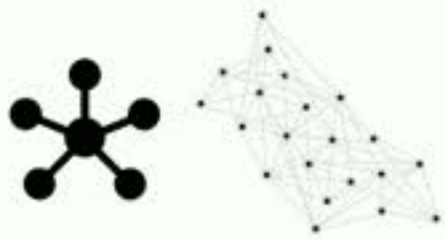
$\forall n.\ M_l(n) \vDash \phi$

Discover[i]

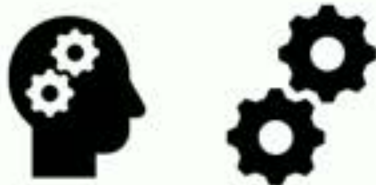
Distributed protocols with consensus components



Abstraction



Parameterized verification



Parameterized synthesis

Specification

ϕ

Process FSM

P

Parameterized verification

Parameterized synthesis

Specification

ϕ

Process FSM

P

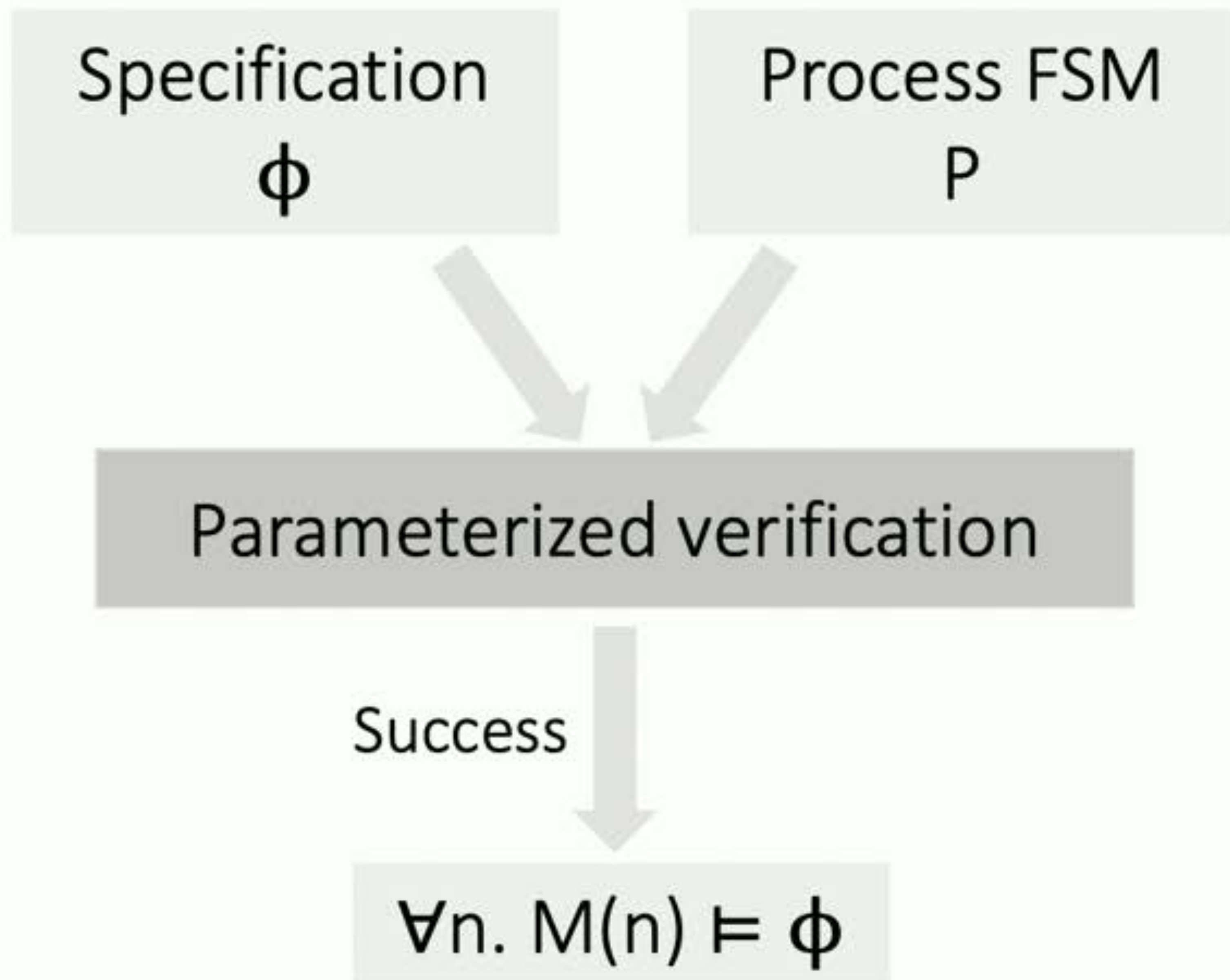
Parameterized verification

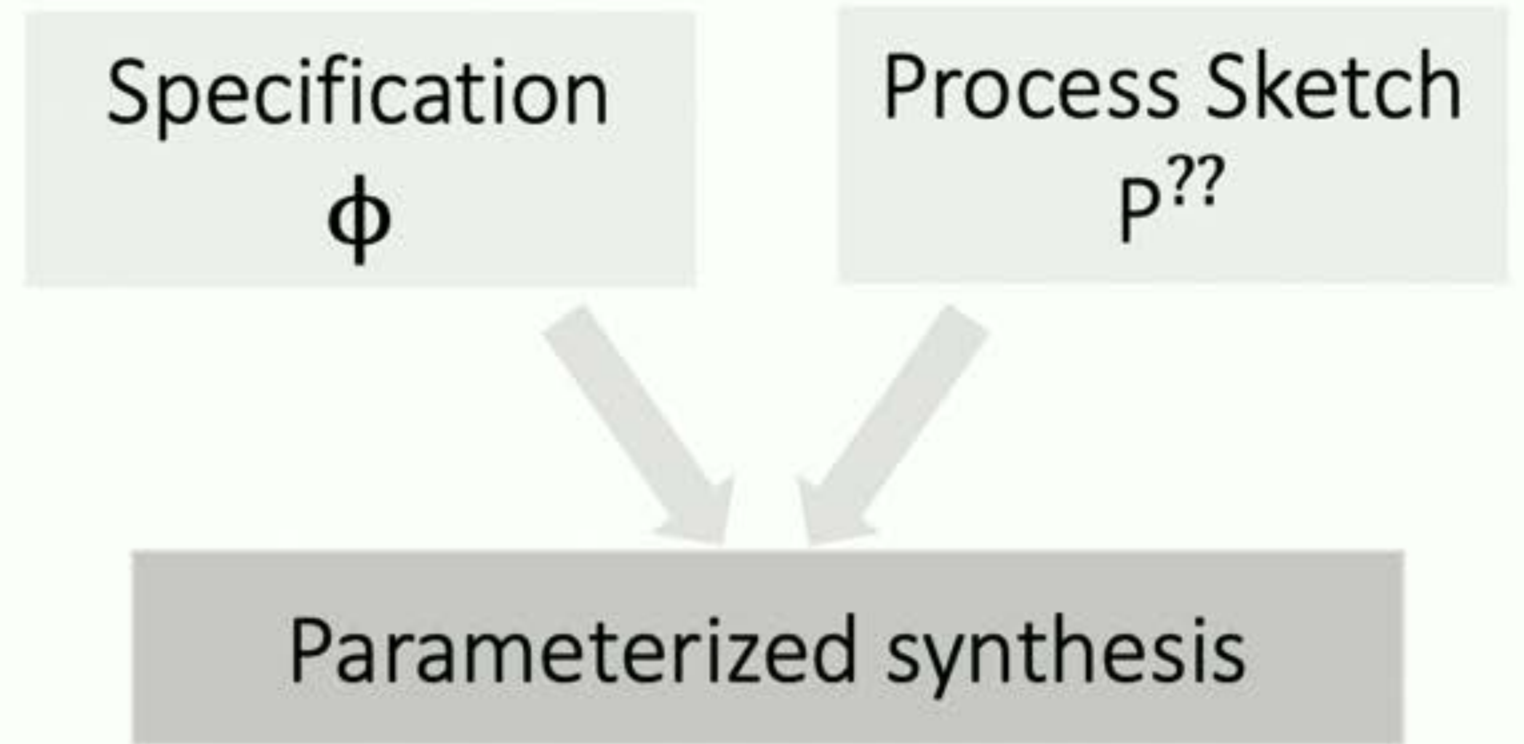
Success

$\forall n.\ M(n) \vDash \phi$

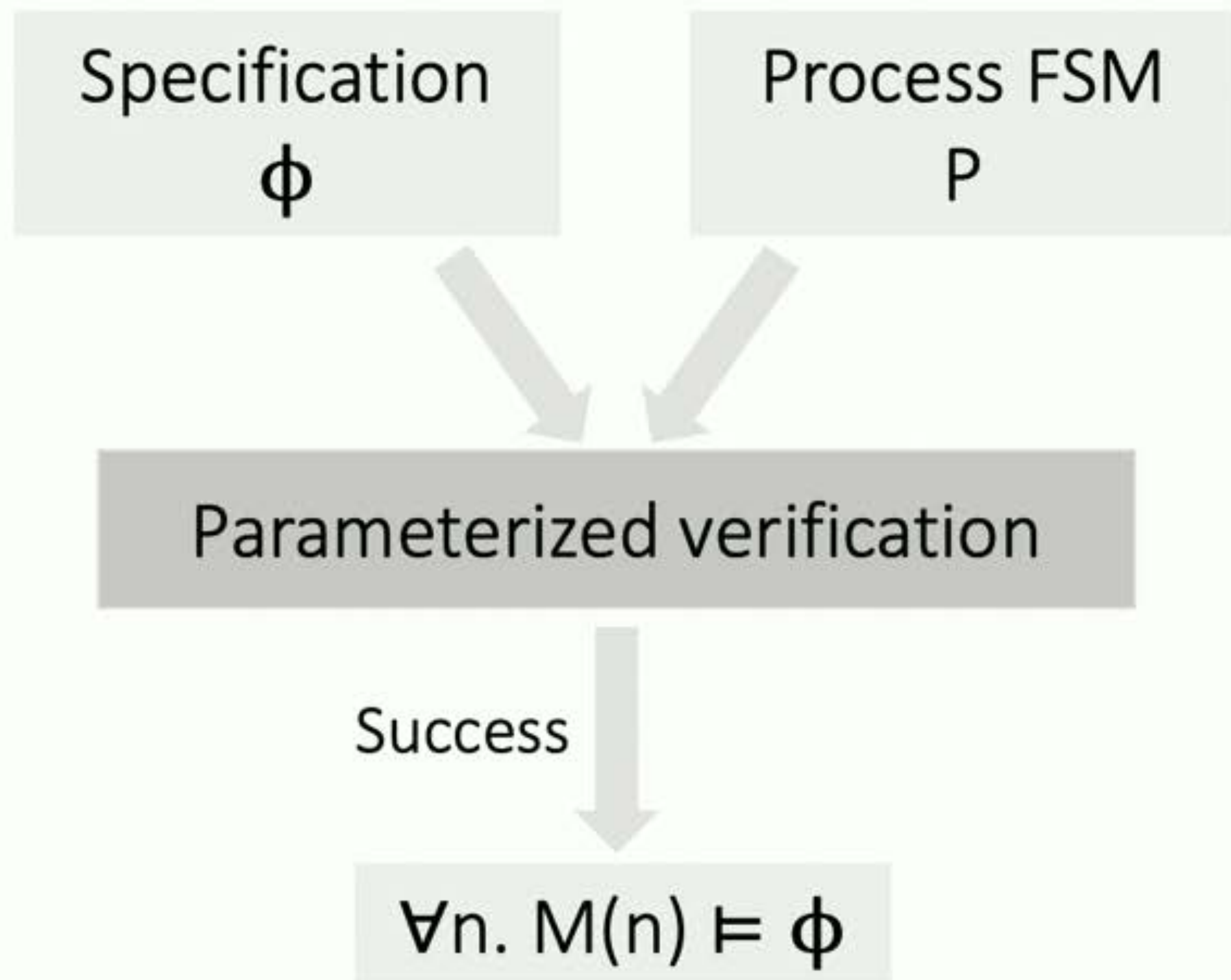
Parameterized synthesis

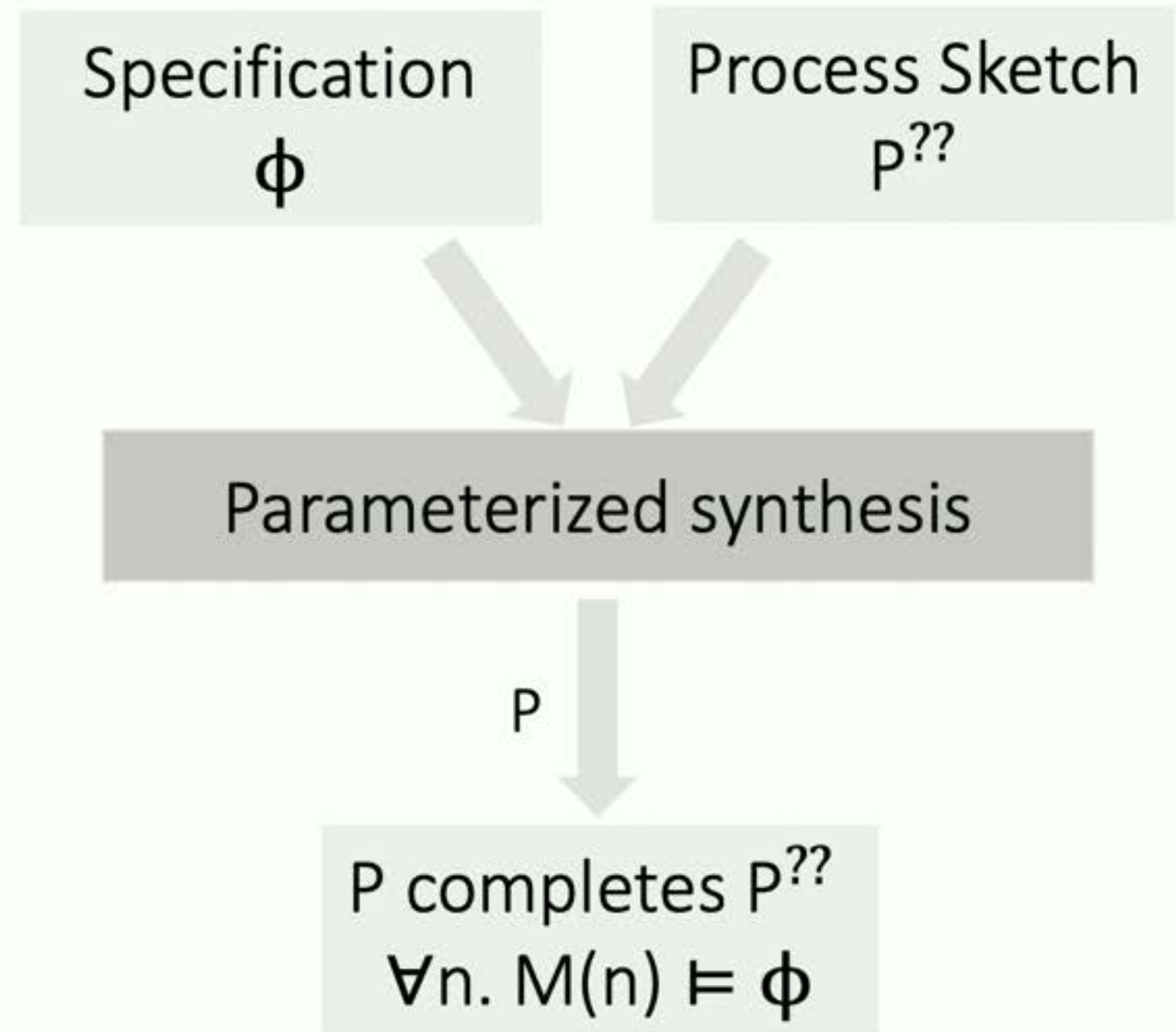
$M(n) = P_1 \parallel \cdots \parallel P_n$



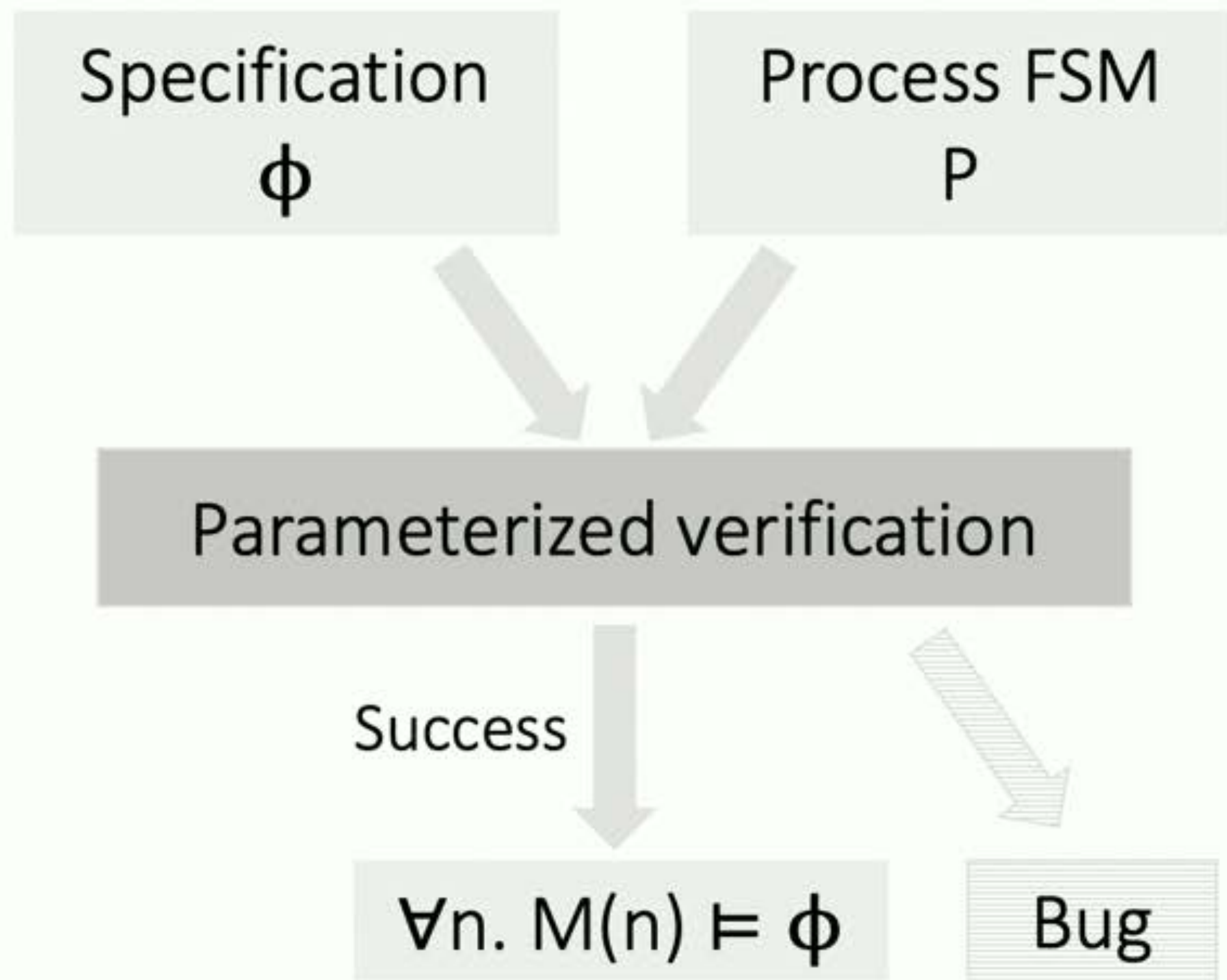


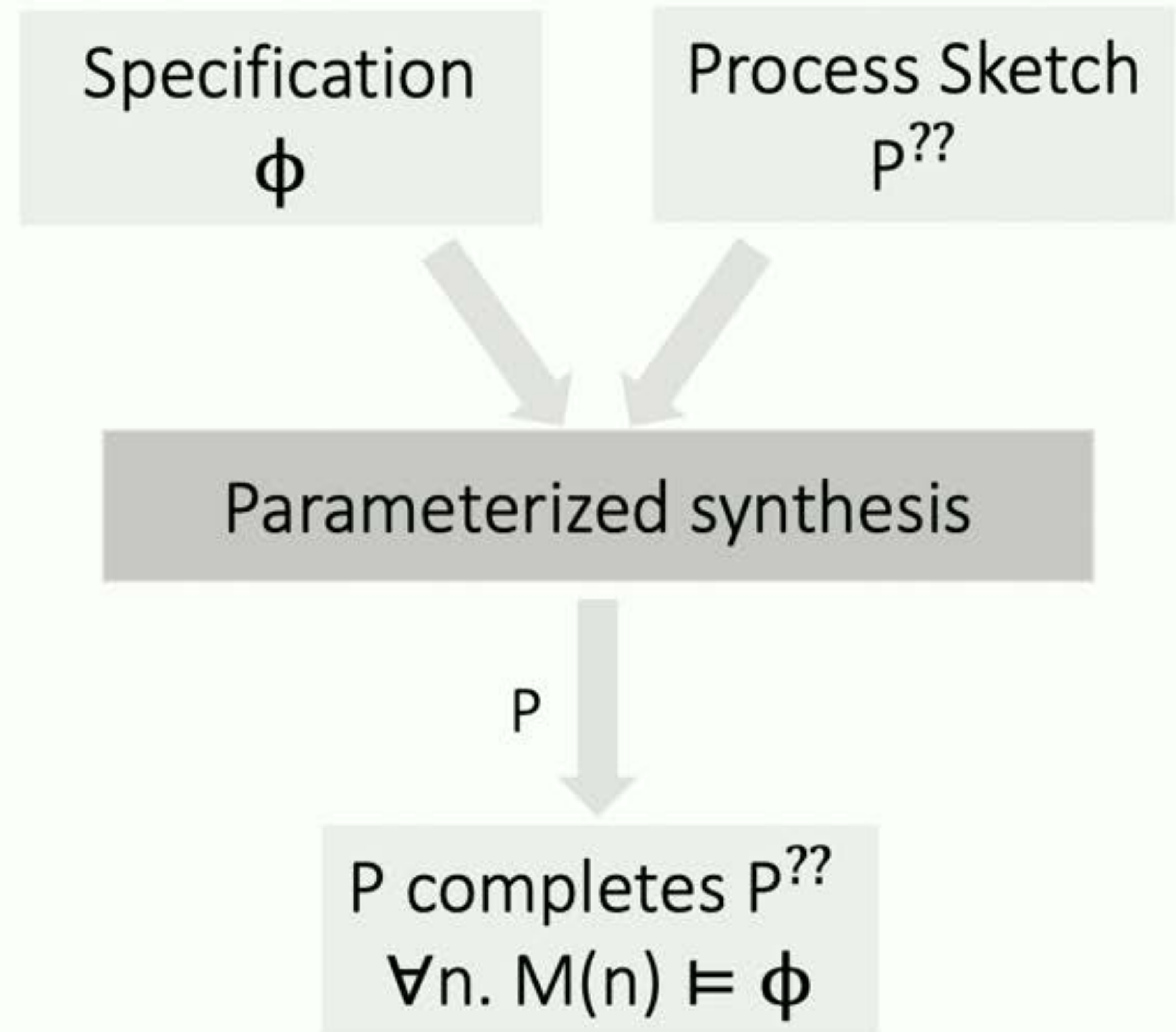
$M(n) = P_1 \| \cdots \| P_n$



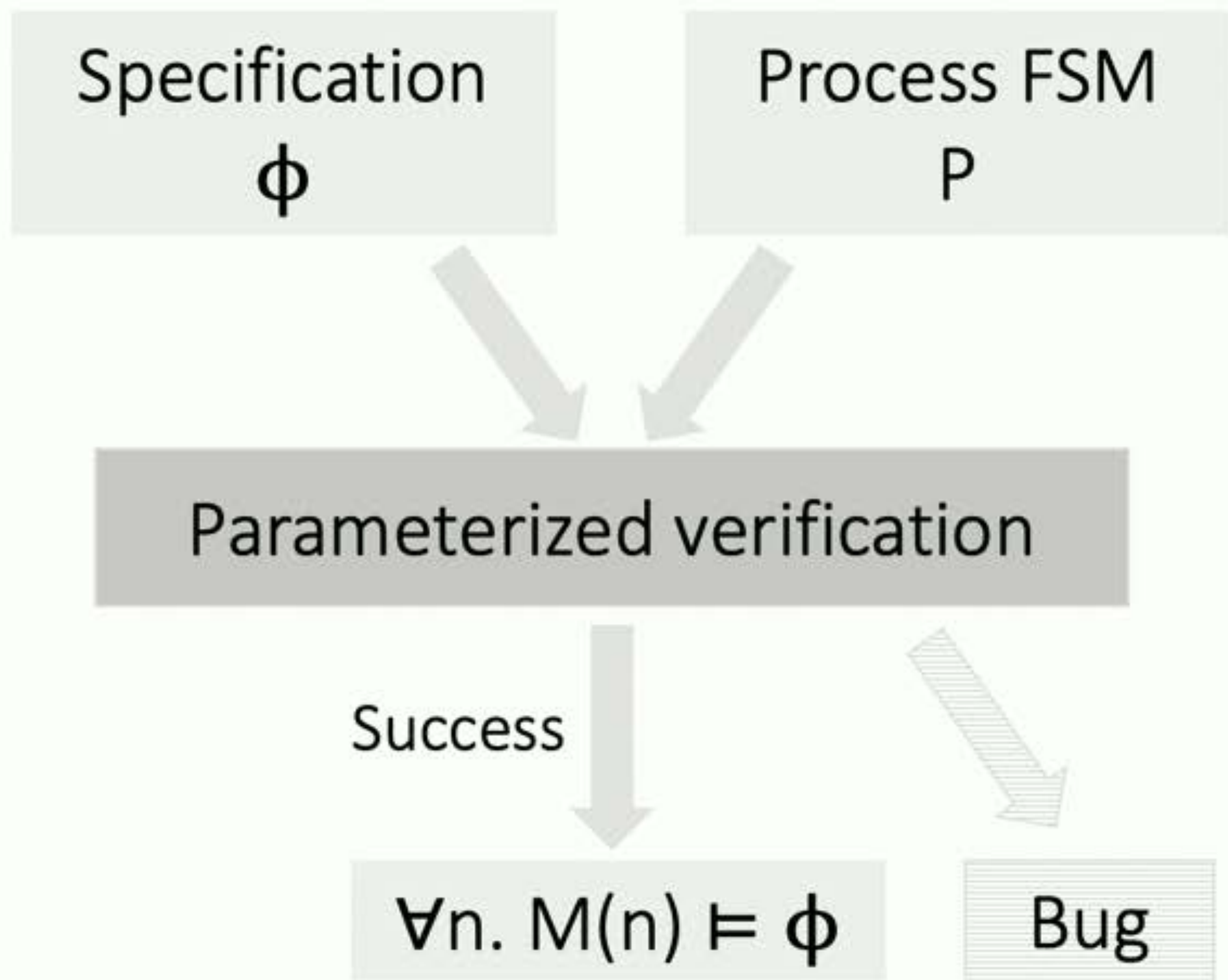


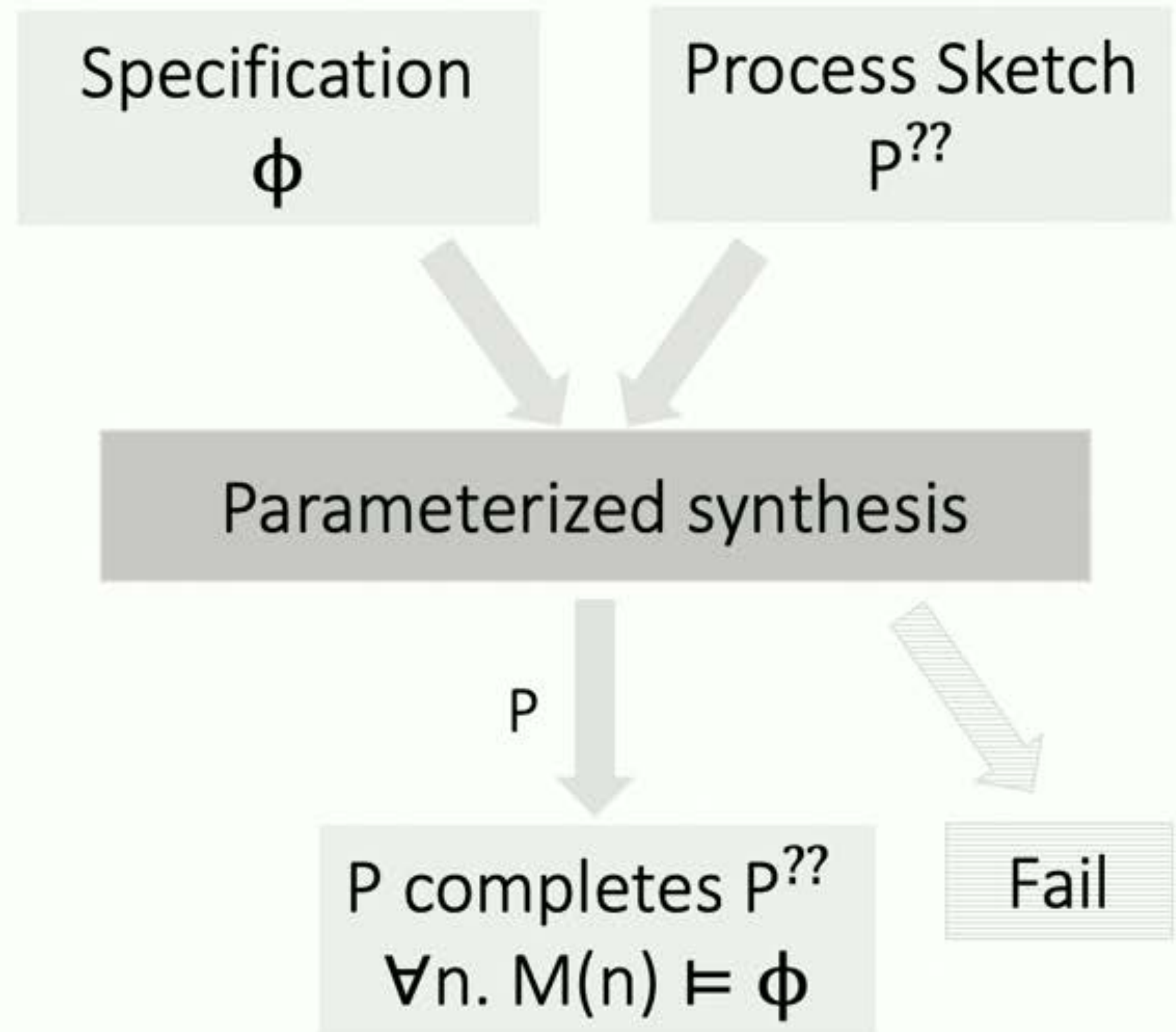
$$M(n) = P_1 \parallel \cdots \parallel P_n$$





$M(n) = P_1 \parallel \cdots \parallel P_n$

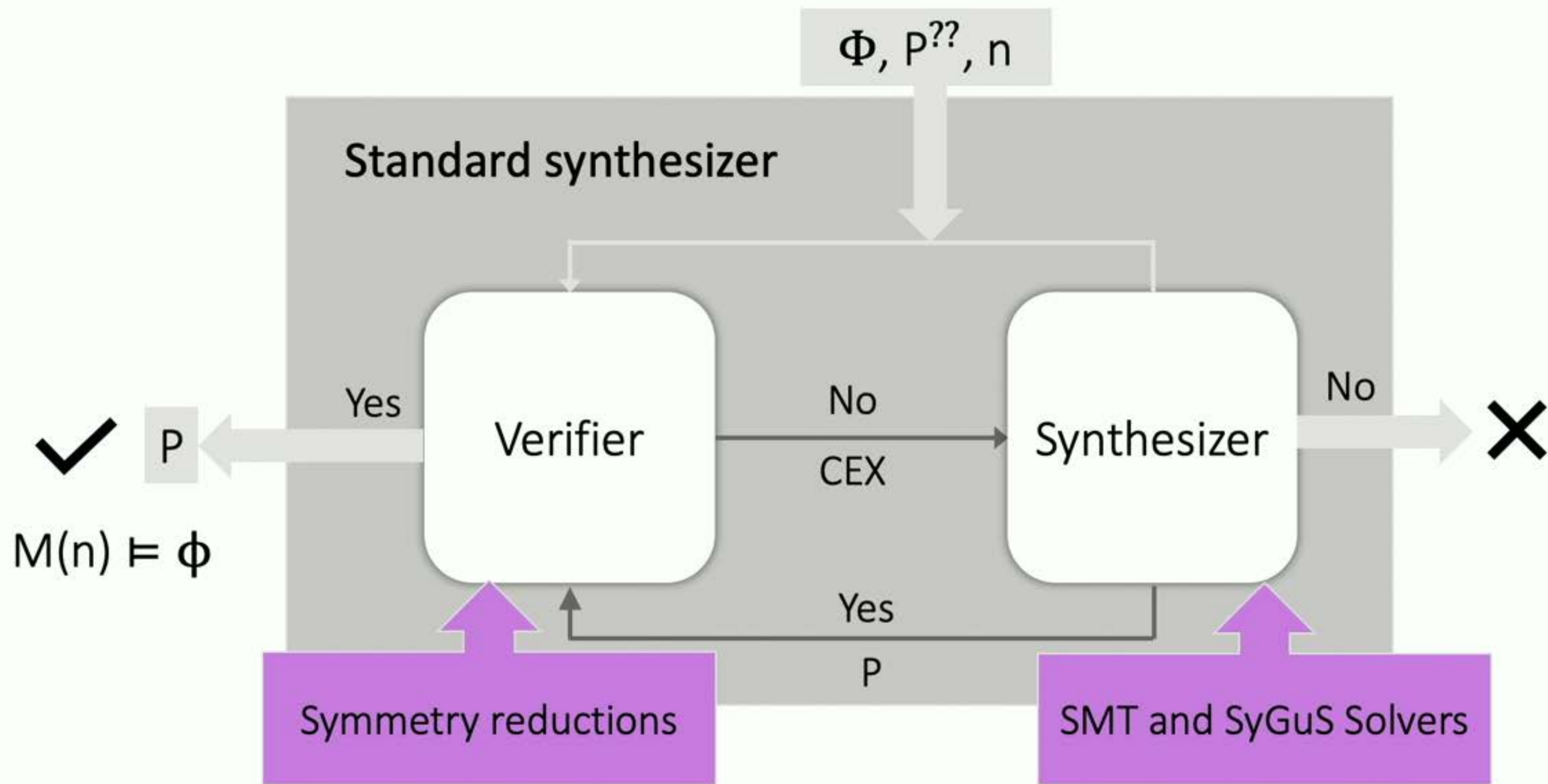




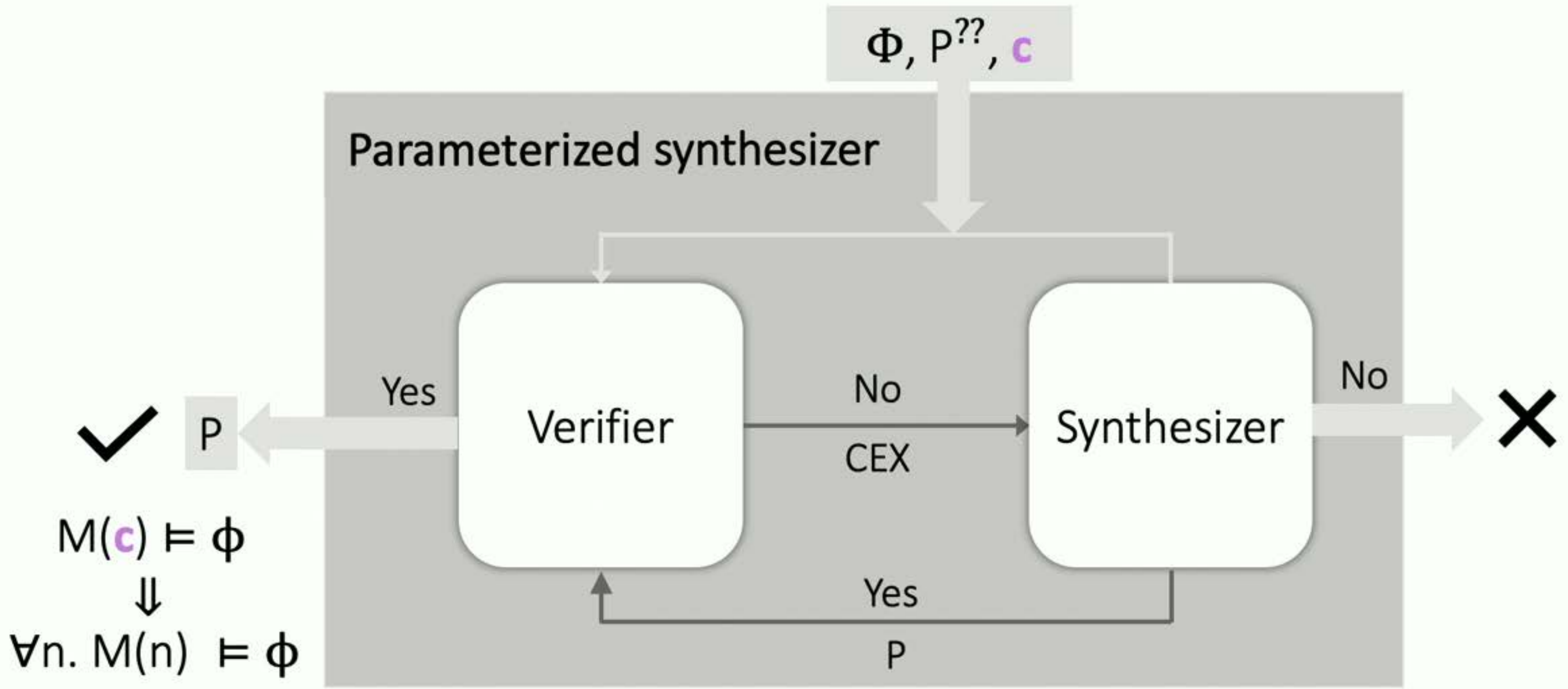
$M(n) = P_1 \parallel \cdots \parallel P_n$

Fixed n: Counterexample-guided inductive synthesis

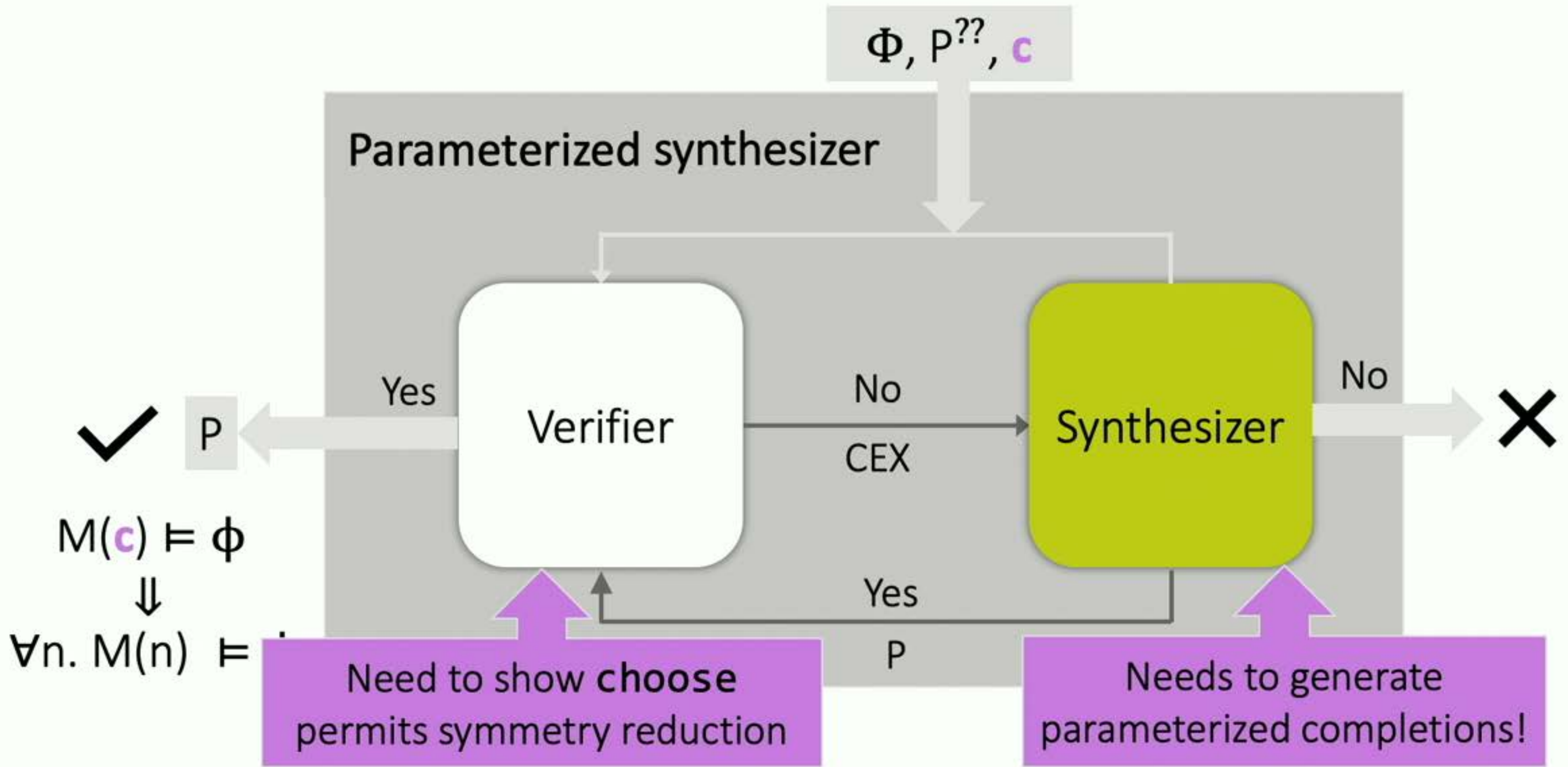
[Alur et al, 2014, 2015]



Arbitrary n: Use cut-off c!



Arbitrary n: Use cut-off c!

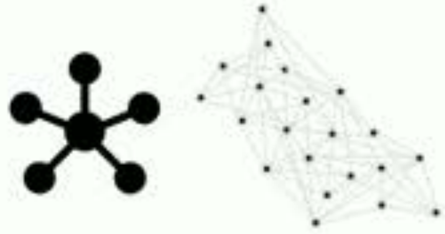


Benchmark	#Local Vars	#Locations	Cut-off	#Iterations	Time(s)
Chubby application	3	9	2	7	4.76
Smoke detector (SD)	7	7	3	49	128.4
SD with reset	7	7	3	31	117.2
SD with 2 **choose**'s	8	8	3	30	144.4
SATS	17	14	5	6	287.6

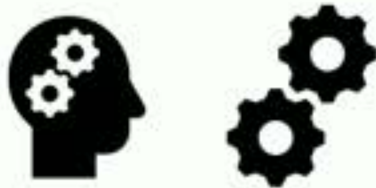
Discover[i]

Distributed protocols with consensus components

Abstraction



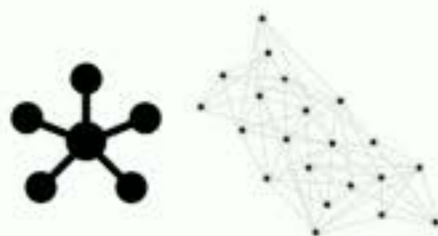
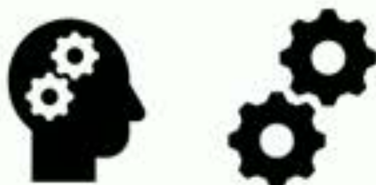
Parameterized verification



Parameterized synthesis

Discover[i]

Distributed protocols with components

Failures		Abstractions
Liveness		Parameterized verification
Other components		Parameterized synthesis