
Youssef Hanna
Iowa State U.


David Samuelson
Iowa State U.


Samik Basu
Iowa State U.

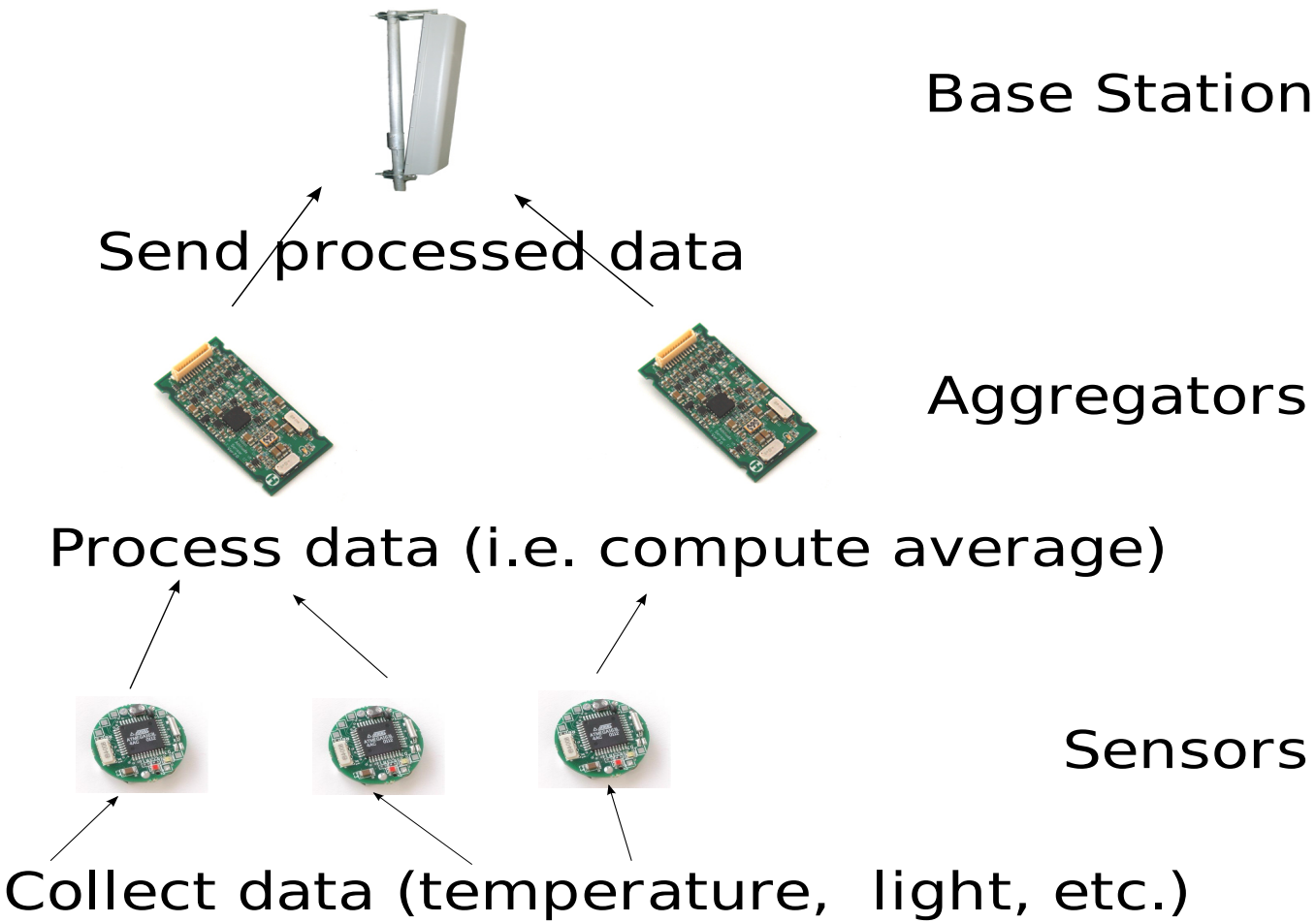

Hridesh Rajan
Iowa State U.

Automating Cut-off for Multi-Parameterized Systems

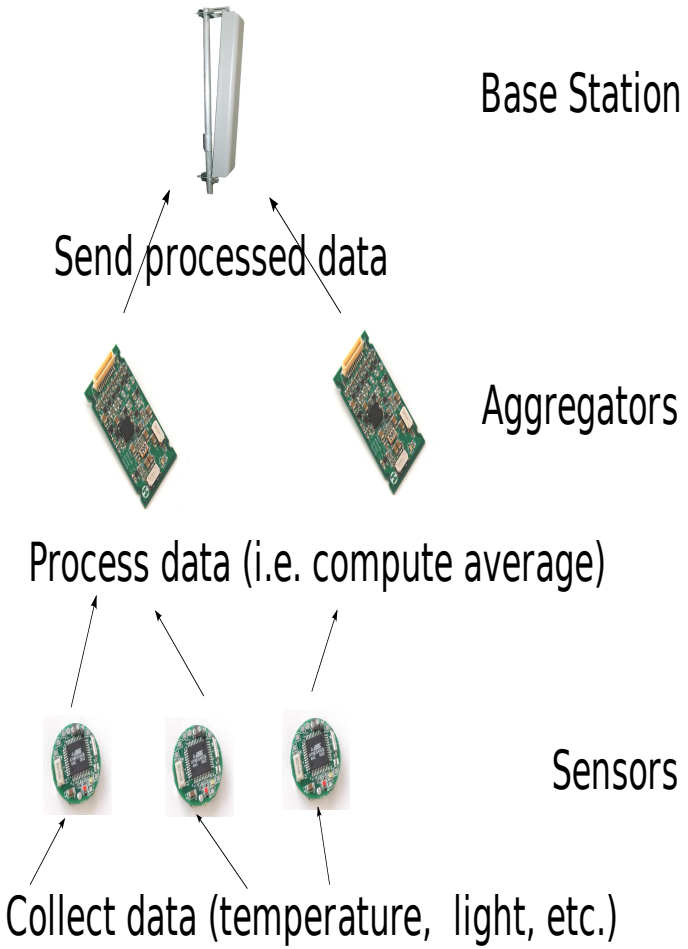
Midwest Verification Day 2010
To appear in the proceedings of the International Conference on Formal Engineering Methods (ICFEM 2010)

September 17, 2010

Sensor Network (Binary Tree)

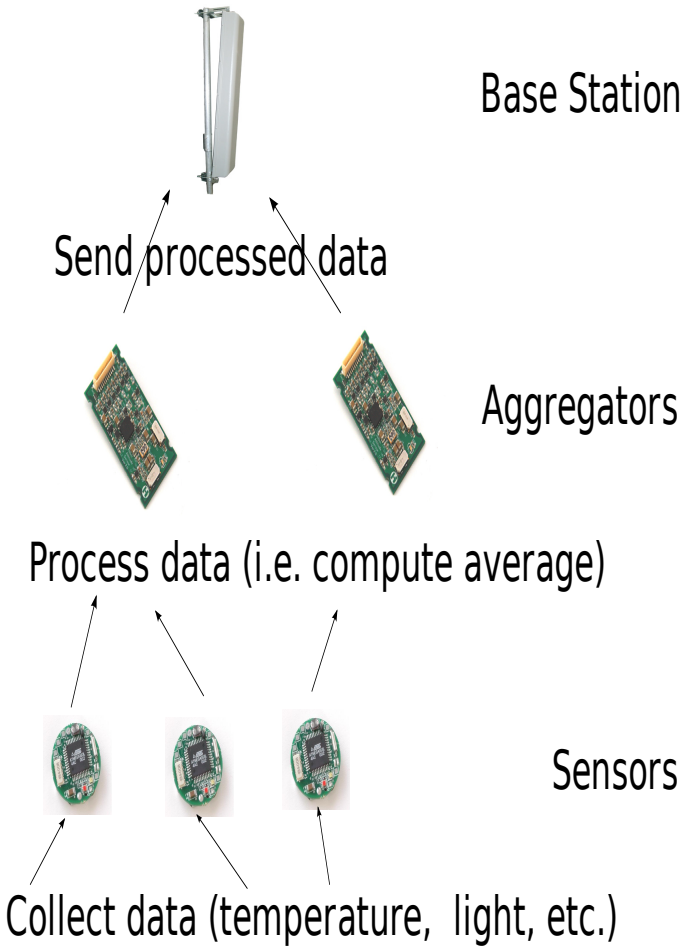


Sensor Network (Binary Tree)



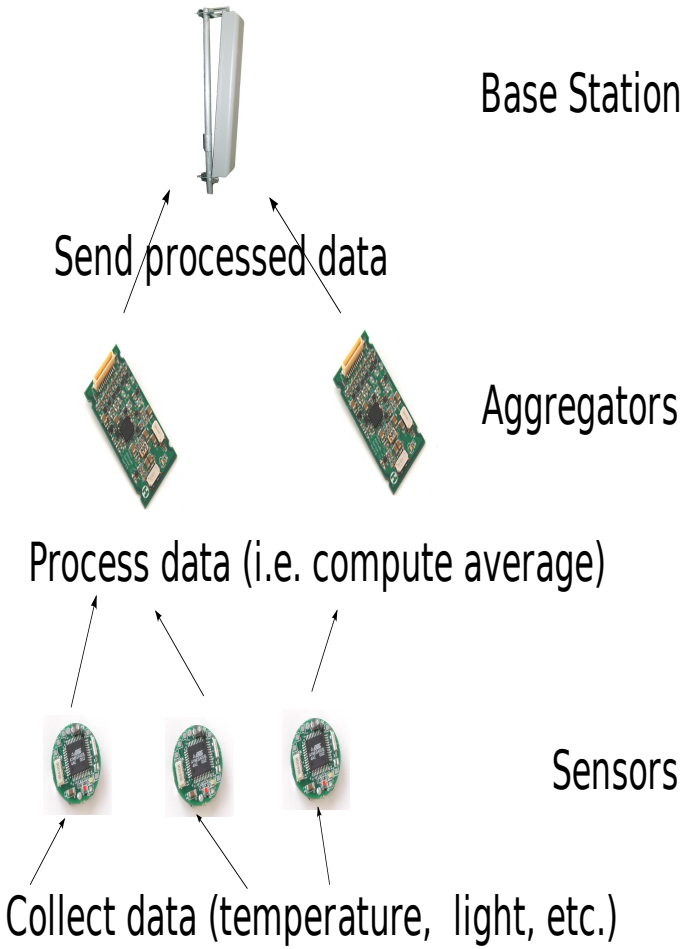
- In this system, there exists:
  - ***m*** Aggregators
  - ***n*** Sensors

Sensor Network (Binary Tree)



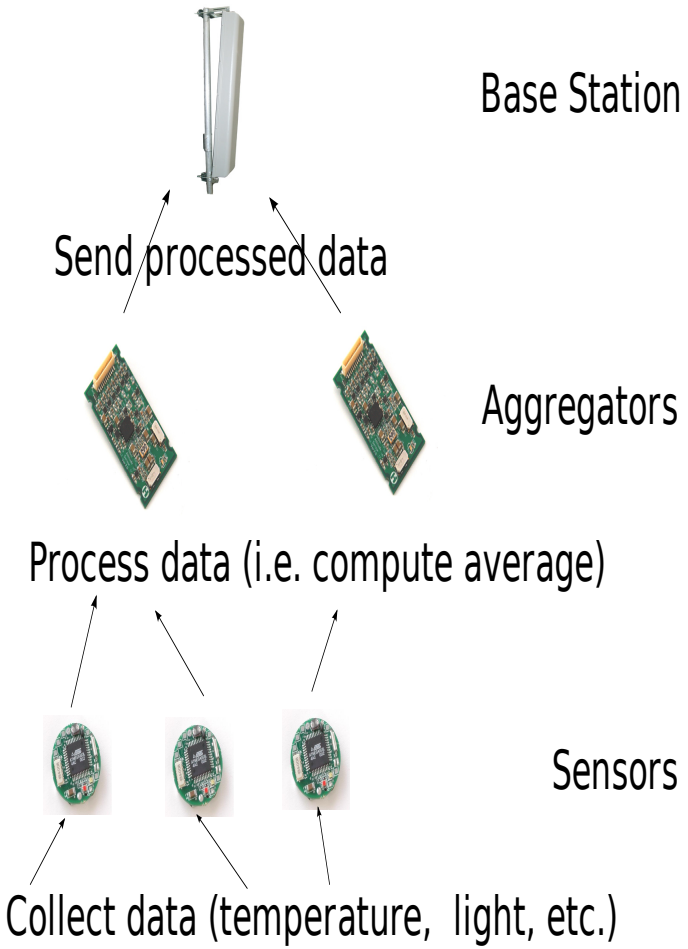
- In this system, there exists:
  - m Aggregators
  - n Sensors
- In this instance of the system:
  - $m = 2$
  - $n = 3$
- This system instance: $sys(2, 3)$

Sensor Network (Binary Tree)



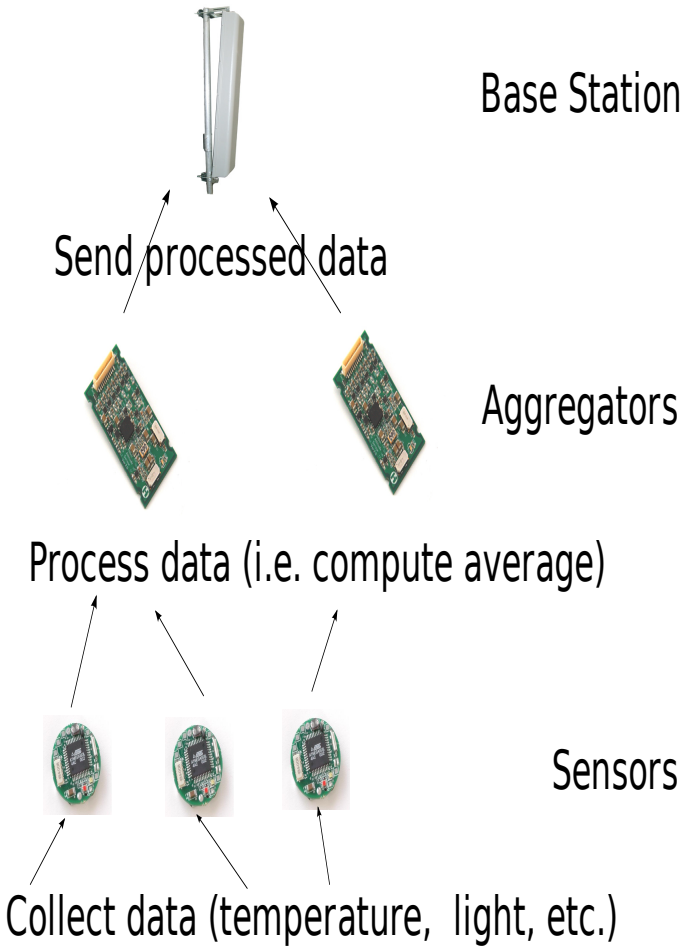
- In this system, there exists:
  - m Aggregators
  - n Sensors
- In this instance of the system:
  - $m = 2$
  - $n = 3$
- This system instance: $sys(2, 3)$
- Two different process types.

Sensor Network (Binary Tree)



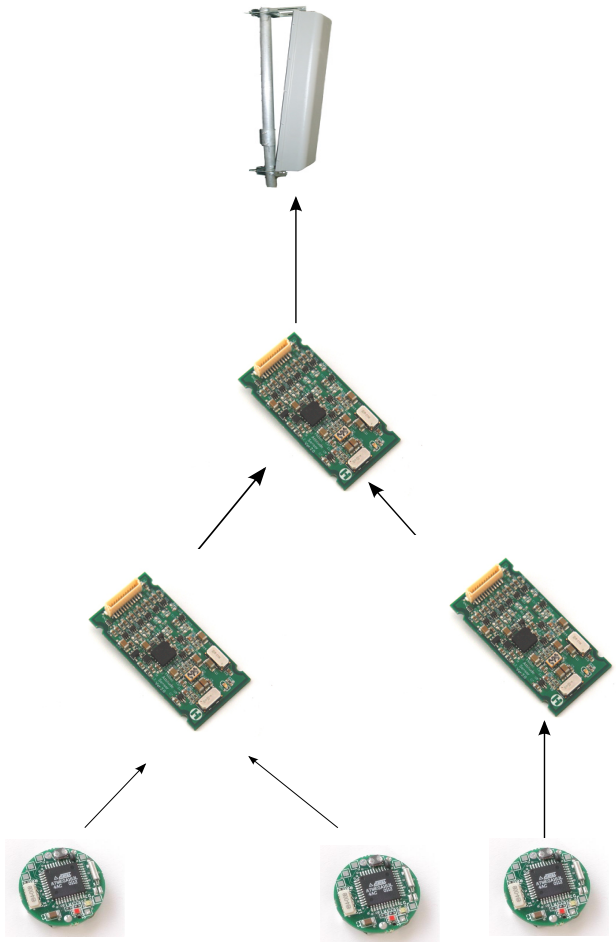
- In this system, there exists:
  - m Aggregators
  - n Sensors
- In this instance of the system:
  - $m = 2$
  - $n = 3$
- This system instance: $sys(2, 3)$
- Two different process types.
- For every process type, there is a parameter.

Verifying a Property



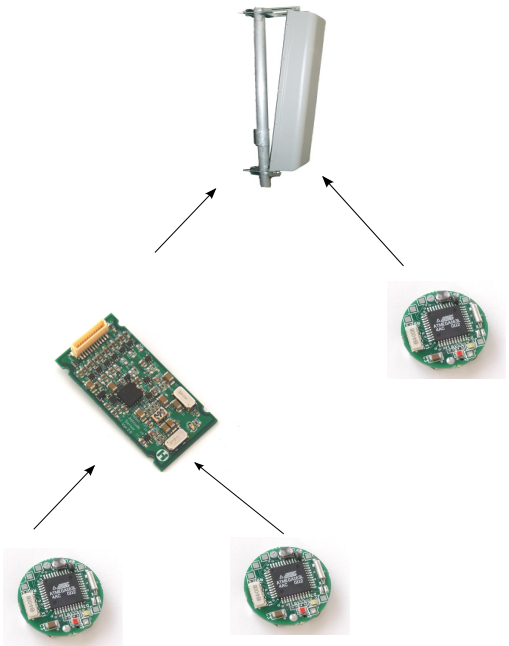
- Example property φ to verify:
  - Base station receives the average of all collected data.
- Which instance to verify?

Verifying a Property



$sys(3, 3)$

Verifying a Property



$sys(1, 3)$

Which Instance to Verify?

- If $\boldsymbol{sys(2,3)} \models \varphi$, will $\boldsymbol{sys(10,15)} \models \varphi$?

Which Instance to Verify?

- If $\boldsymbol{sys(2,3)} \models \varphi$, will $\boldsymbol{sys(10,15)} \models \varphi$?
- If $\boldsymbol{sys(m,n)} \models \varphi$, will $\boldsymbol{sys(m+1,n+1)} \models \varphi$?

Verifying Parameterized Systems

- **Problem of Verifying Multi-Parameterized Systems:**
  - Given a system $sys(n_1, n_2, \ldots, n_t)$ with t different types of processes, where n_i is the parameter for process type i,

[1] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett., 22(6):307-309, 1986.

Verifying Parameterized Systems

- **Problem of Verifying Multi-Parameterized Systems:**
  - Given a system $sys(n_1, n_2, \ldots, n_t)$ with t different types of processes, where n_i is the parameter for process type i,
  - given a property φ,

[1] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett., 22(6):307-309, 1986.

Verifying Parameterized Systems

- **Problem of Verifying Multi-Parameterized Systems:**
  - Given a system $sys(n_1, n_2, \ldots, n_t)$ with t different types of processes, where n_i is the parameter for process type i,
  - given a property φ,
  - **is the property satisfied for *every* instance of the system?**

[1] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett., 22(6):307-309, 1986.

Verifying Parameterized Systems

- **Problem of Verifying Multi-Parameterized Systems:**
  - Given a system $sys(n_1, n_2, \ldots, n_t)$ with t different types of processes, where n_i is the parameter for process type i,
  - given a property φ,
  - **is the property satisfied for *every* instance of the system?**
- For systems with only one parameterized process type (i.e. $sys(n)$), this is an undecidable problem! [1].

[1] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett., 22(6):307-309, 1986.

Existing Work

- Large amount of existing work (mostly for **1** parameter)[2] [3].
- Key idea based on the notion of *cut-off*.
- Identify a number $\boldsymbol{k}$ s.t.

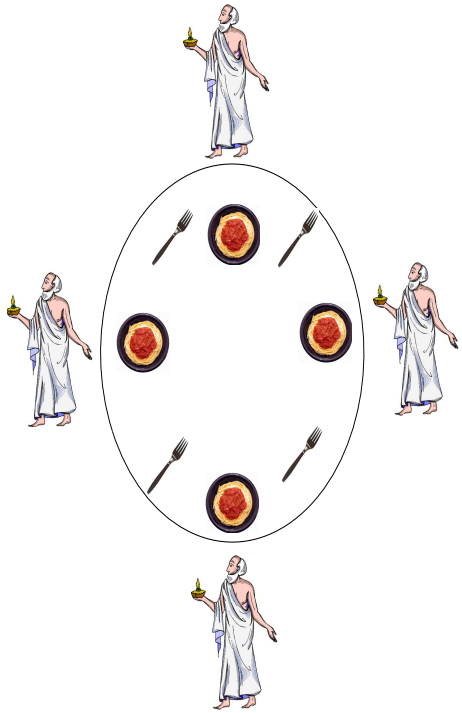
$$\boldsymbol{sys(k)} \models \varphi \Leftrightarrow \forall \boldsymbol{n} > \boldsymbol{k} : \boldsymbol{sys(n)} \models \varphi$$

- No need to verify properties for $\boldsymbol{n} > \boldsymbol{k}$
- $\boldsymbol{k}$ is called the *cut-off*

[2] E. A. Emerson, R. J. Trefler, and T. Wahl. Reducing model checking of the few to the one. In ICFEM, pp. 94-113, 2006.

[3] C. N. Ip and D. L. Dill. Verifying systems with replicated components in murφ. In CAV, pages 147-158, 1996.

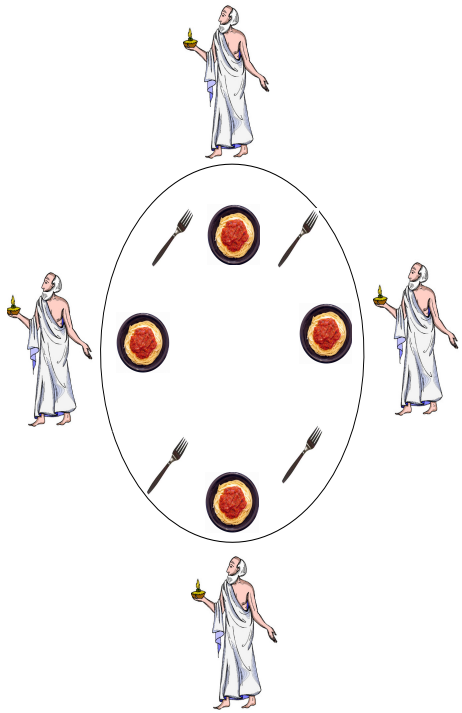
Cut-off for Single-parameterized Systems



Dining Philosopher

- Shared fork between every two philosophers
- Property φ: two neighbor philosophers cannot eat together

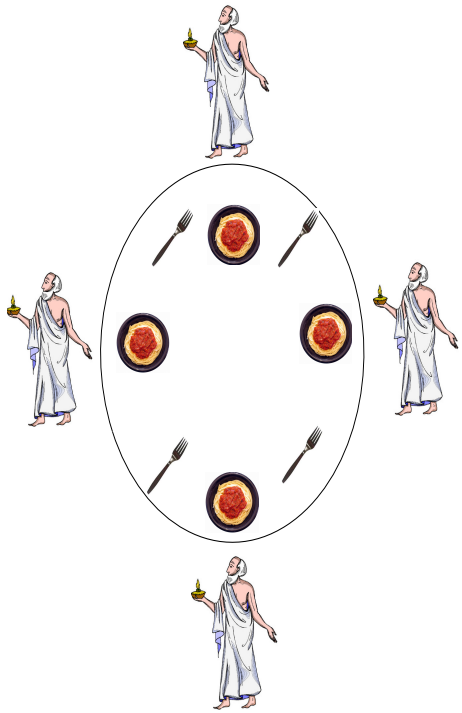
Cut-off for Single-parameterized Systems



φ: two neighbor philosophers cannot eat together

- Cut-off is **4** [Emerson and Cahlon TACAS 2002]
- We showed cut-off is **3** [Hanna, Y., Basu, S., and Rajan, H. ESEC/FSE 2009]

Cut-off for Single-parameterized Systems



φ: two neighbor philosophers cannot eat together

- Cut-off is **4** [Emerson and Cahlon TACAS 2002]
- We showed cut-off is **3** [Hanna, Y., Basu, S., and Rajan, H. ESEC/FSE 2009]
- $\boldsymbol{sys(3)} \models \varphi \Leftrightarrow \forall \boldsymbol{n} > \mathbf{3}\ \boldsymbol{sys(n)} \models \varphi$

Problem with Existing Work

- Most ideas focus on a specific system with a single parameter

[4] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In POPL, pages 85-94, 1995

Problem with Existing Work

- Most ideas focus on a specific system with a single parameter
  - [[4]] focuses on ring systems with a *single* parameter.

[4] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In POPL, pages 85-94, 1995

Problem with Existing Work

- Most ideas focus on a specific system with a single parameter
  - [4] focuses on ring systems with a *single* parameter.
  - Not immediately applicable to new systems with
    - different topologies.
    - different number of parameters.
  - . . . without developing new theories from first principles.

[4] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In POPL, pages 85-94, 1995

Problem with Existing Work

- [5] is the first to consider parameterized systems with different types processes.

[5] Emerson, E. A. and Kahlon, V. 2002. Model Checking Large-Scale and Parameterized Resource Allocation Systems. In TACAS (2002).

Problem with Existing Work

- [5] is the first to consider parameterized systems with different types processes.
  - works only for specific systems where processes communicate through shared resources.

[5] Emerson, E. A. and Kahlon, V. 2002. Model Checking Large-Scale and Parameterized Resource Allocation Systems. In TACAS (2002).

Proposed Solution

A Language-based Technique for Cut-off Computation [6]

Technical Contributions:

A representation strategy to specify system as input,
which enables
a cut-off generation algorithm .

Key Benefits:

- Fully automated – Ease of verification.
- Topology and parameter independent – User Specified

[6] Hanna, Y., Basu, S., and Rajan, H.. Behavioral Automata Composition for Automatic Topology Independent Verification of Parameterized Systems. ESEC/FSE 2009

Key Idea

1. **Input**

Key Idea

1. **Input**
   - all possible actions of a process for every process type.
   - system topology

Key Idea

1. **Input**
   - all possible actions of a process for every process type.
   - system topology
2. **Generate**

Key Idea

1. **Input**
   - all possible actions of a process for every process type.
   - system topology
2. **Generate**
   - the maximal behavior that a process of every type can induce in *any* environment, and

Key Idea

1. **Input**
   - all possible actions of a process for every process type.
   - system topology
2. **Generate**
   - the maximal behavior that a process of every type can induce in *any* environment, and
3. **Find**

Key Idea

1. **Input**
   - all possible actions of a process for every process type.
   - system topology
2. **Generate**
   - the maximal behavior that a process of every type can induce in *any* environment, and
3. **Find**
   - the smallest system instance that allows such behaviors to happen, then **the size of this instance is the cut-off.**

Key Idea

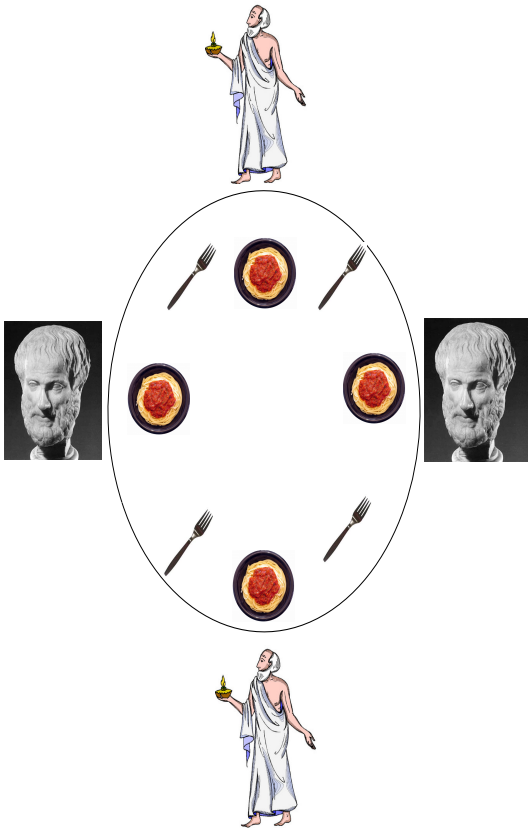
1. **Input**
   - all possible actions of a process for every process type.
   - system topology
2. **Generate**
   - the maximal behavior that a process of every type can induce in *any* environment, and
3. **Find**
   - the smallest system instance that allows such behaviors to happen, then **the size of this instance is the cut-off.**

- *Any larger system cannot exhibit any more behavior.*

Outline

- Multi-Parameterized System Definition
- Problem Statement
- **Solution:**
  - Define all possible actions of a process of every type.
  - Define system topology.
  - Generate maximal behavior a process of every type can induce in *any* environment.
  - Find smallest system allowing such behaviors to be exhibited.
- Case Studies

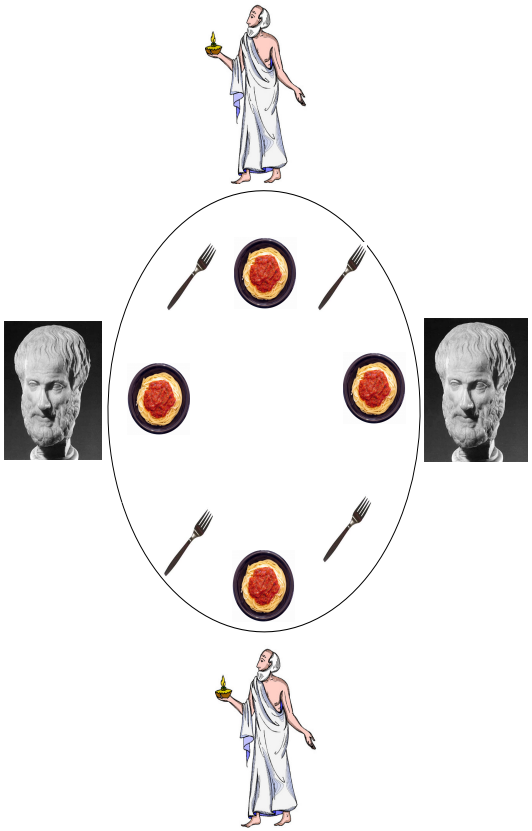
Right-Left Dining Philosophers (RLDP)



In Emerson and Cahlon
TACAS 2002

- Type “Left” phil.
  - Grab their left fork first
- Type “Right” phil.
  - Grab their right fork first

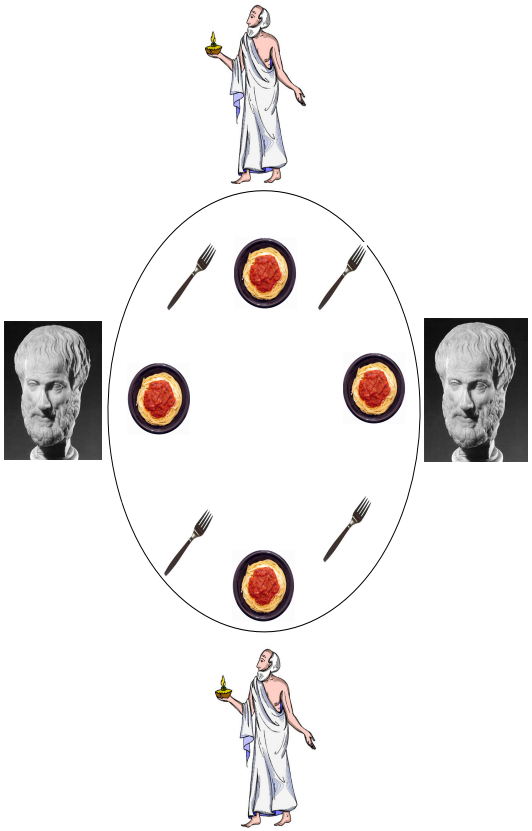
Right-Left Dining Philosophers (RLDP)



In Emerson and Cahlon
TACAS 2002

- Type “Left” phil.
  - Grab their left fork first
- Type “Right” phil.
  - Grab their right fork first
- A parameter exists for every type

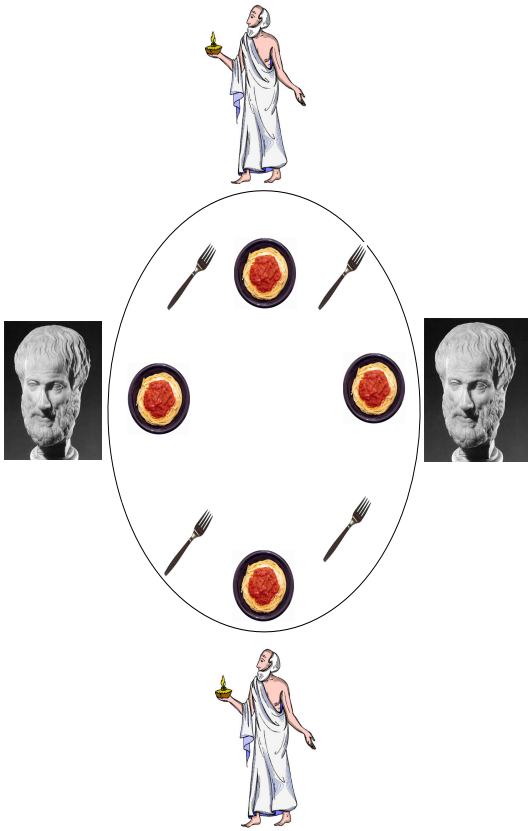
Right-Left Dining Philosophers (RLDP)



In Emerson and Cahlon TACAS 2002

- Type “Left” phil.
  - Grab their left fork first
- Type “Right” phil.
  - Grab their right fork first
- A parameter exists for every type
- φ: No two philosophers can eat at the same time

Right-Left Dining Philosophers (RLDP)



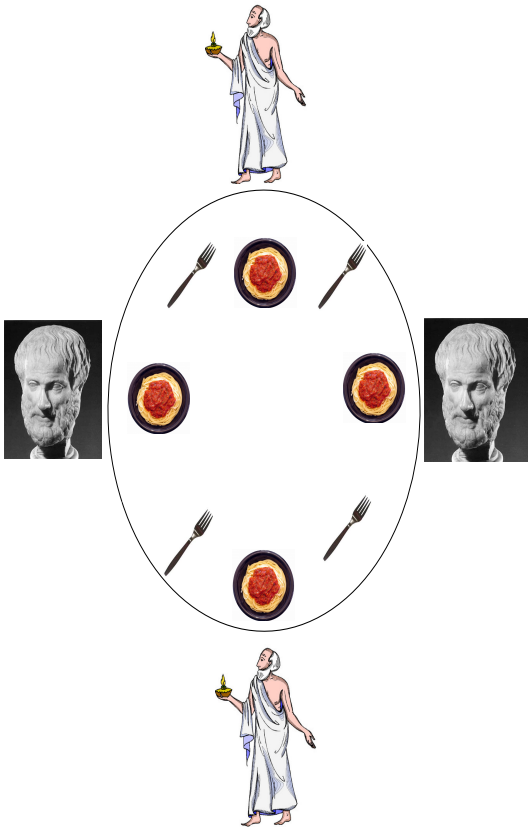
In Emerson and Cahlon TACAS 2002

- Type “Left” phil.
  - Grab their left fork first
- Type “Right” phil.
  - Grab their right fork first
- A parameter exists for every type
- φ: No two philosophers can eat at the same time
- **Q: Which Instance to verify?**

Steps for Our Technique

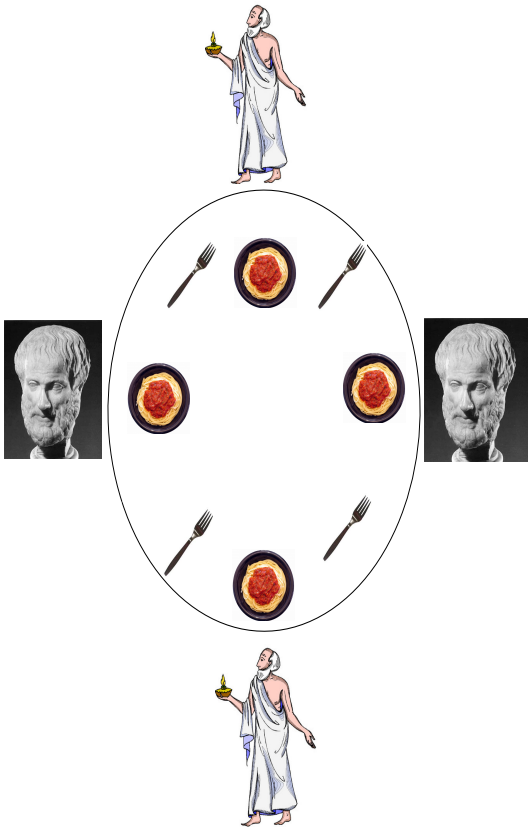
1. **Define actions of a process of every type.**
2. Define system topology.
3. Generate maximal behavior a process of every type can generate in *any* environment.
4. Find smallest system that exhibits *all* generated maximal behaviors.

“Left” Philosopher Actions



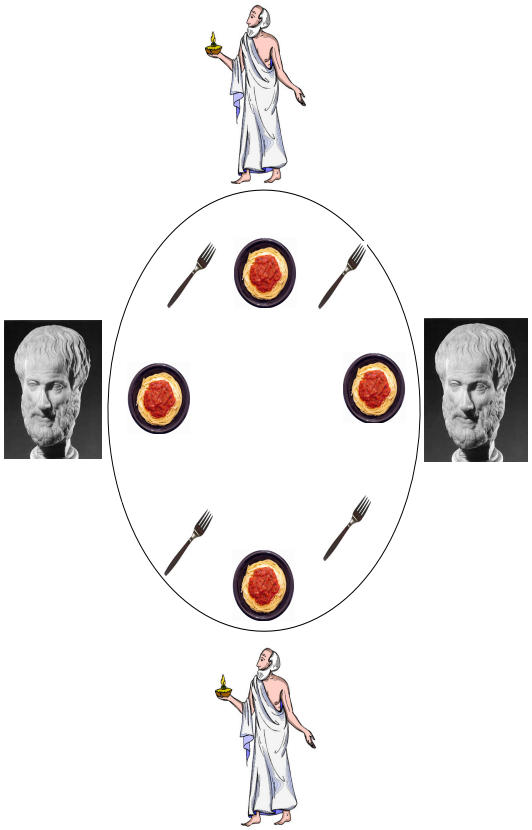
- **Send** request neighbor for left fork.

“Left” Philosopher Actions



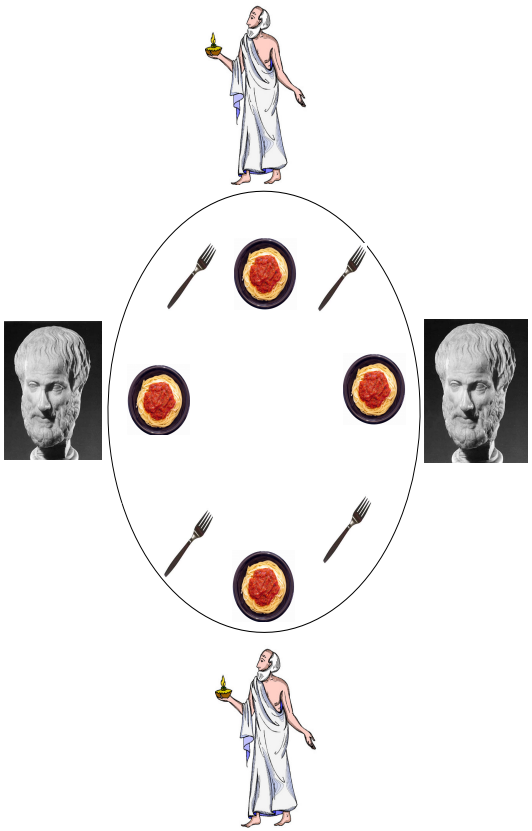
- **Send** request neighbor for left fork.
- **Receive** approval to acquire left fork then **Send** request for right fork.

“Left” Philosopher Actions



- **Send** request neighbor for left fork.
- **Receive** approval to acquire left fork then **Send** request for right fork.
- **Receive** rejection then **Send** another request.
- . . .

“Left” Philosopher Actions



- **Send** request neighbor for left fork.
- **Receive** approval to acquire left fork then **Send** request for right fork.
- **Receive** rejection then **Send** another request.
- . . .

A similar set of actions exists for “Right” philosopher type (requesting right fork first)

“Left” Philosopher Actions as Behavioral Automata

1. **Send** request neighbor for left fork.

“Left” Philosopher Behavioral Automata

1. `L-ASKL: [neating, epsilon]->[waitl, askl2]`

“Left” Philosopher Actions as Behavioral Automata

1. **Send** request neighbor for left fork.
2. **Receive** approval to acquire left fork then **Send** request for right fork.

“Left” Philosopher Behavioral Automata

1. `L-ASKL: [neating, epsilon]->[waitl, askl2]`
2. `L-ASK: [waitl, lfree] -> [waitr, askr2]`

“Left” Philosopher Actions as Behavioral Automata

1. **Send** request neighbor for left fork.
2. **Receive** approval to acquire left fork then **Send** request for right fork.
3. **Receive** rejection then **Send** another request. . . .

“Left” Philosopher Behavioral Automata

1. `L-ASKL: [neating, epsilon]->[waitl, askl2]`
2. `L-ASK: [waitl, lfree] -> [waitr, askr2]`
3. `L-ReASKL: [waitl, ltaken] -> [waitl, askl2]`

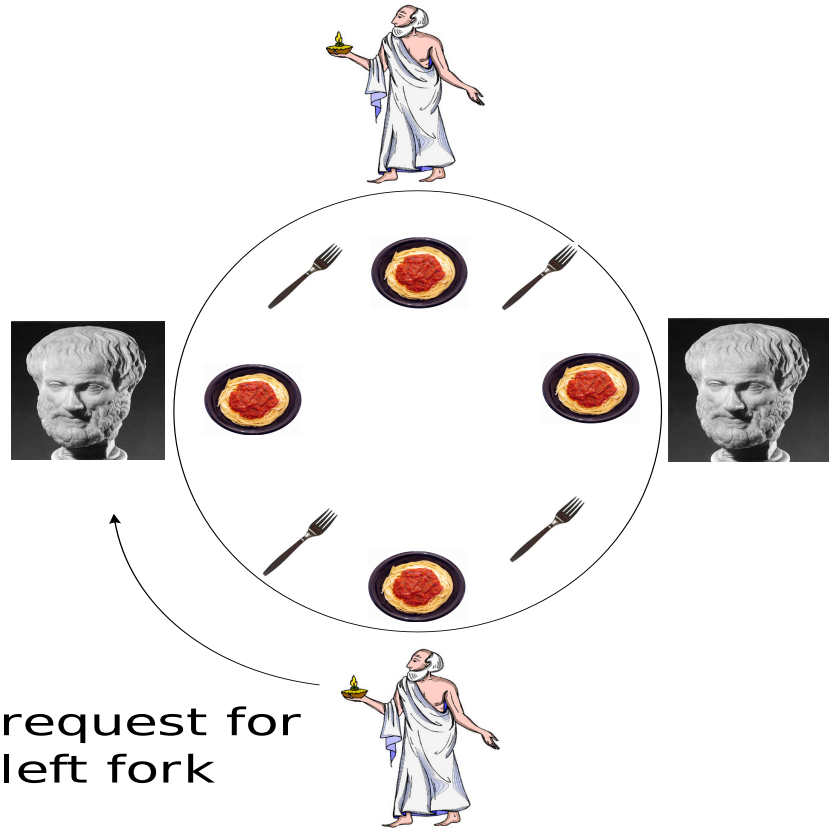
   `...`

Steps for Our Technique

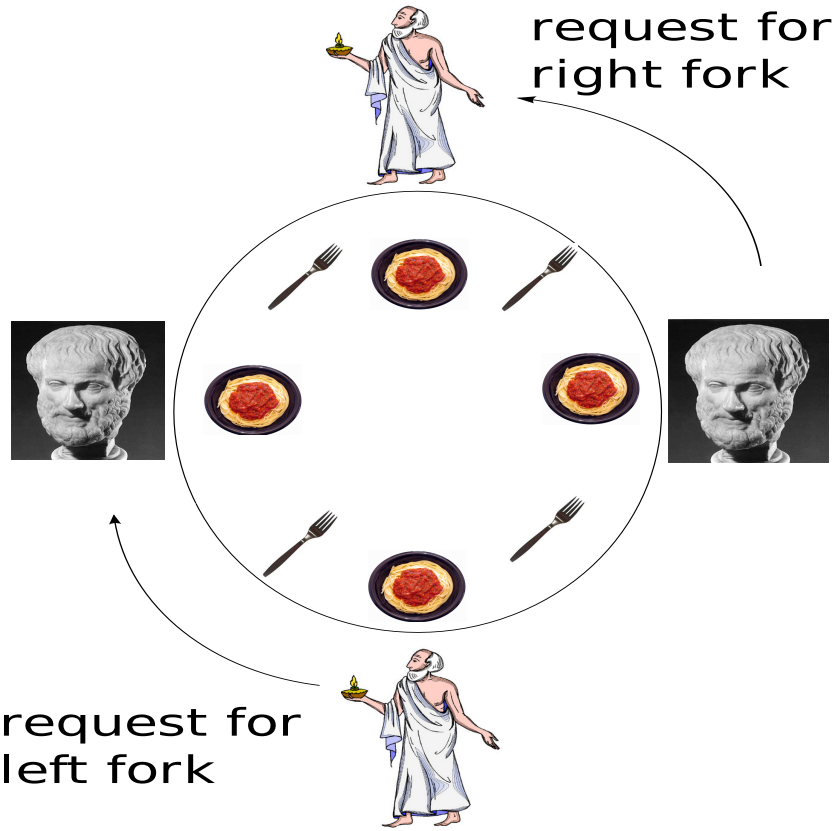
1. Define actions of a process.
2. **Define system topology**.
3. Generate maximal behavior a process of every type can generate in *any* environment.
4. Find smallest system that exhibits *all* generated maximal behaviors.

System Topology

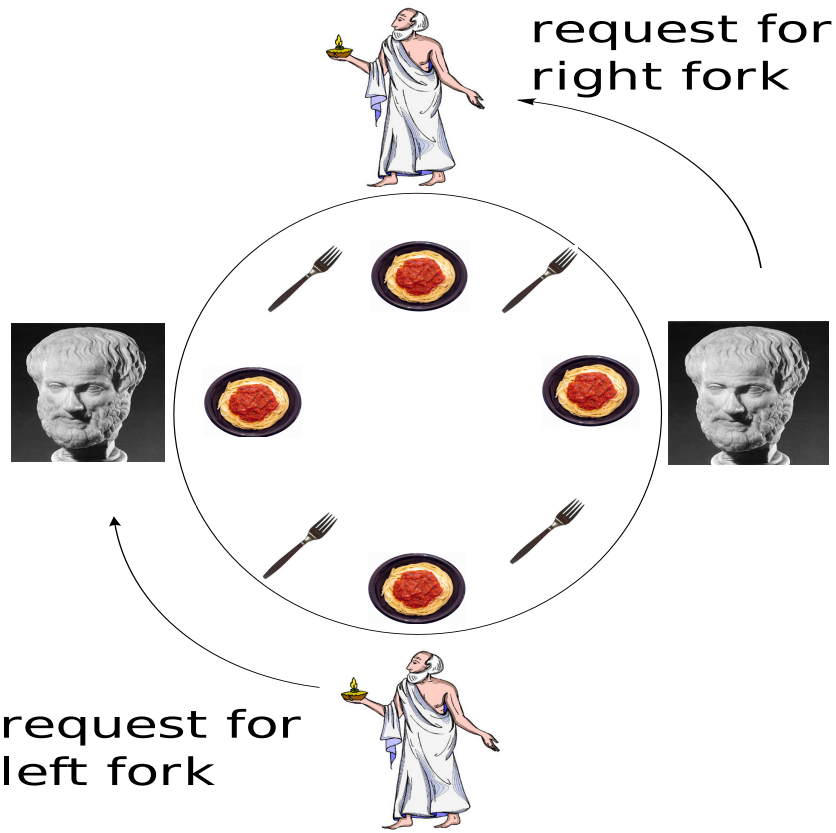
Define how processes are connected together



System Topology



System Topology



- Will connect processes in the form of a ring

Steps for Our Technique

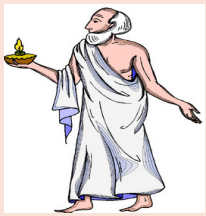
1. Define actions of a process.
2. Define system topology.
3. **Generate maximal behavior a process of every type can generate in *any* environment.**
4. Find smallest system that exhibits *all* generated maximal behaviors.

Maximal Behavior Induced by a “Left” Philosopher

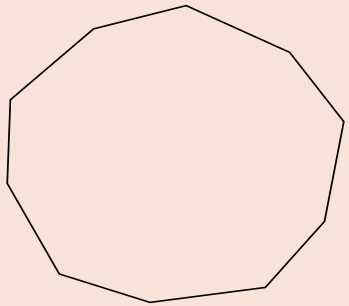
“Left” philosopher initial action

```
L-ASKL: [neating, epsilon] -> [waitl, askl2]
```

Maximal Behavior a “Left” philosopher can induce



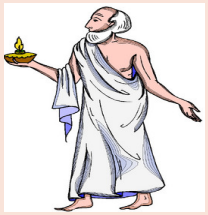
request
left fork



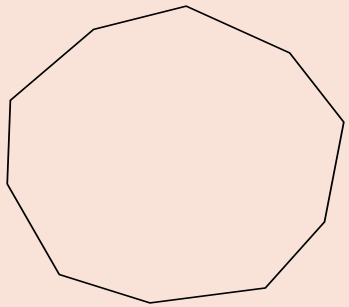
environment

Maximal Behavior Induced by a “Left” Philosopher

Maximal Behavior a “Left” philosopher can induce



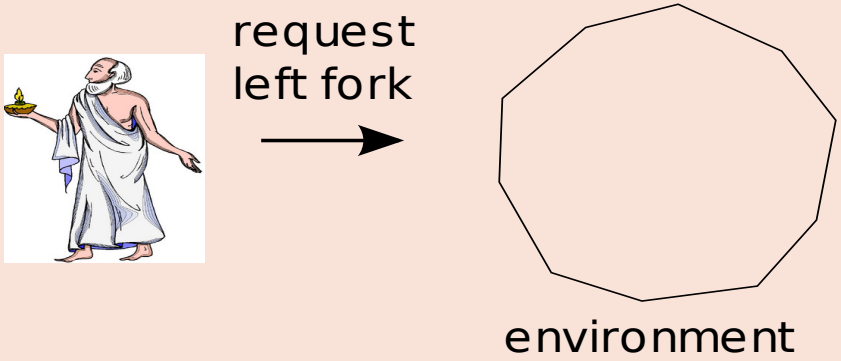
request
left fork



Q: What is the maximal behavior that can happen as a result of that request?

Maximal Behavior Induced by a “Left” Philosopher

Maximal Behavior a “Left” philosopher can induce



Q: What is the maximal behavior that can happen as a result of that request?

A: Check *all* actions that can respond to such request

Maximal Behavior Induced by a “Left” Philosopher

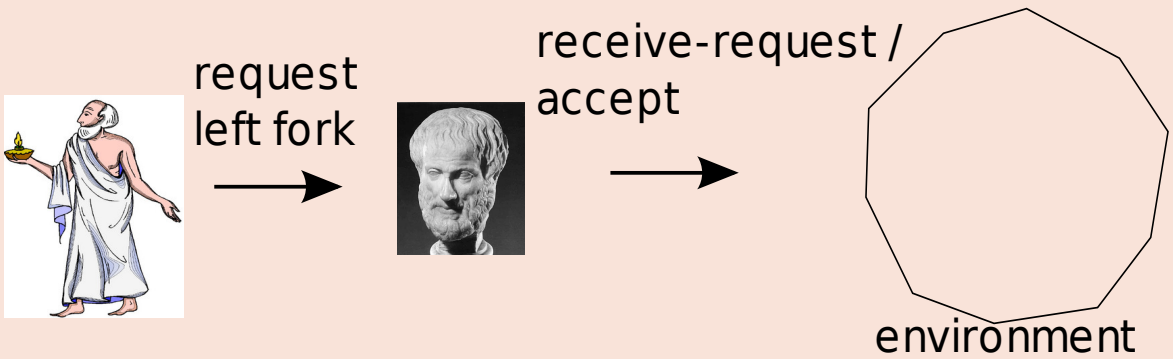
Maximal Behavior a “Left” philosopher can induce

```
L-ASKL: [neating, begin] -> [waitl, askl2]
```

“Right” Philosopher

```
R-FREEL-NE: [neating, askl2]->[neating, lfree]
```

Maximal Behavior a “Left” philosopher can induce



Maximal Behavior Induced by a “Left” Philosopher

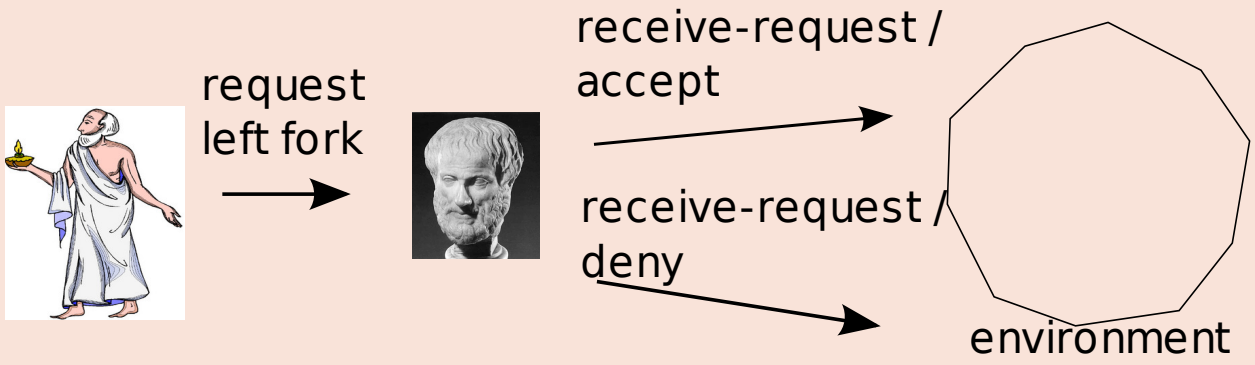
“Left” Philosopher

```
L-ASKL: [neating, epsilon] -> [waitl, askl2]
```

“Right” Philosopher

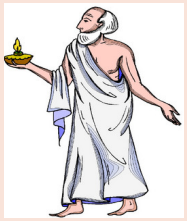
```
R-BUSYL-WR: [waitr, askl2] -> [waitr, ltaken]
```

Maximal Behavior a “Left” philosopher can induce

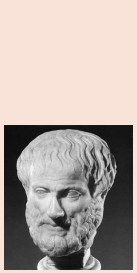


Maximal Behaviors in the RLDP Protocol

Maximal Behavior a “Left” philosopher can induce



Maximal Behavior a “Right” philosopher can induce



Steps for Our Technique

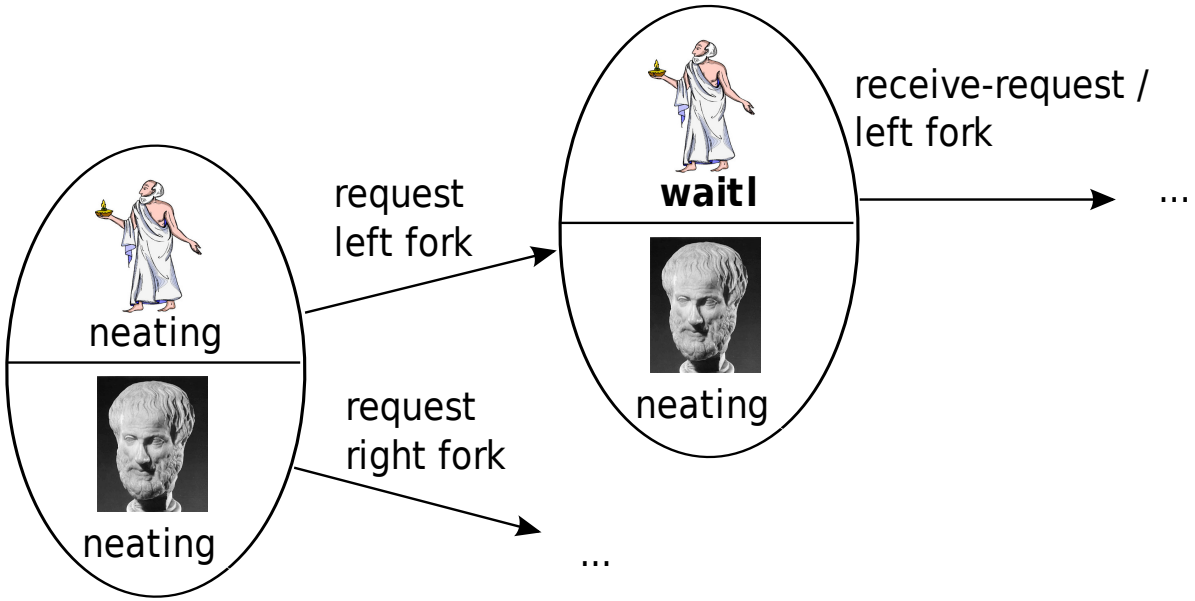
1. Define actions of a process.
2. Define system topology.
3. Generate maximal behavior a process of every type can generate in *any* environment.
4. **Find smallest system that exhibits *all* generated maximal behaviors.**

Goal of Finding Smallest System Instance

- Find the smallest instance that can exhibit all generated maximal behaviors. The size of this system instance $(k_1, k_2, \ldots, k_t)$ is the cut-off

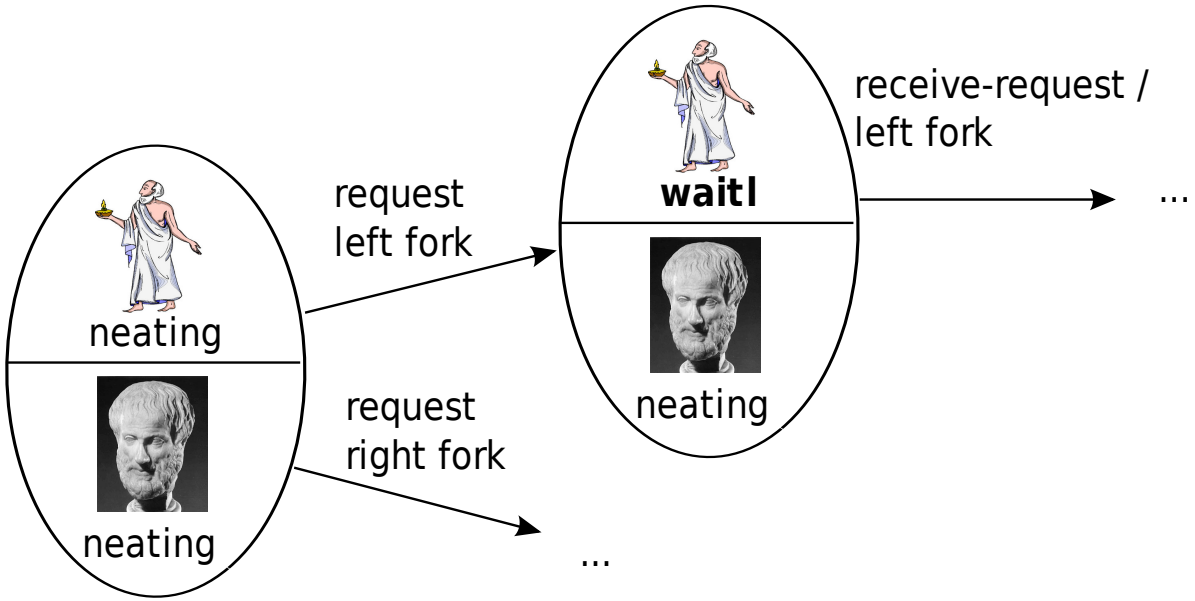
$sys(k_1, k_2, \ldots, k_t)$ satisfies $\varphi \Leftrightarrow$ All larger system instances satisfy φ

System with One Philosopher of Every Type



- **Goal:** The system should have all the sequence of paths in every generated maximal behavior.

System with One Philosopher of Every Type



- **Goal:** The system should have all the sequence of paths in every generated maximal behavior.
- $sys(1_l, 1_r)$ does not all the sequence of paths, it is not the cut-off

Cut-off for RLDP Protocol

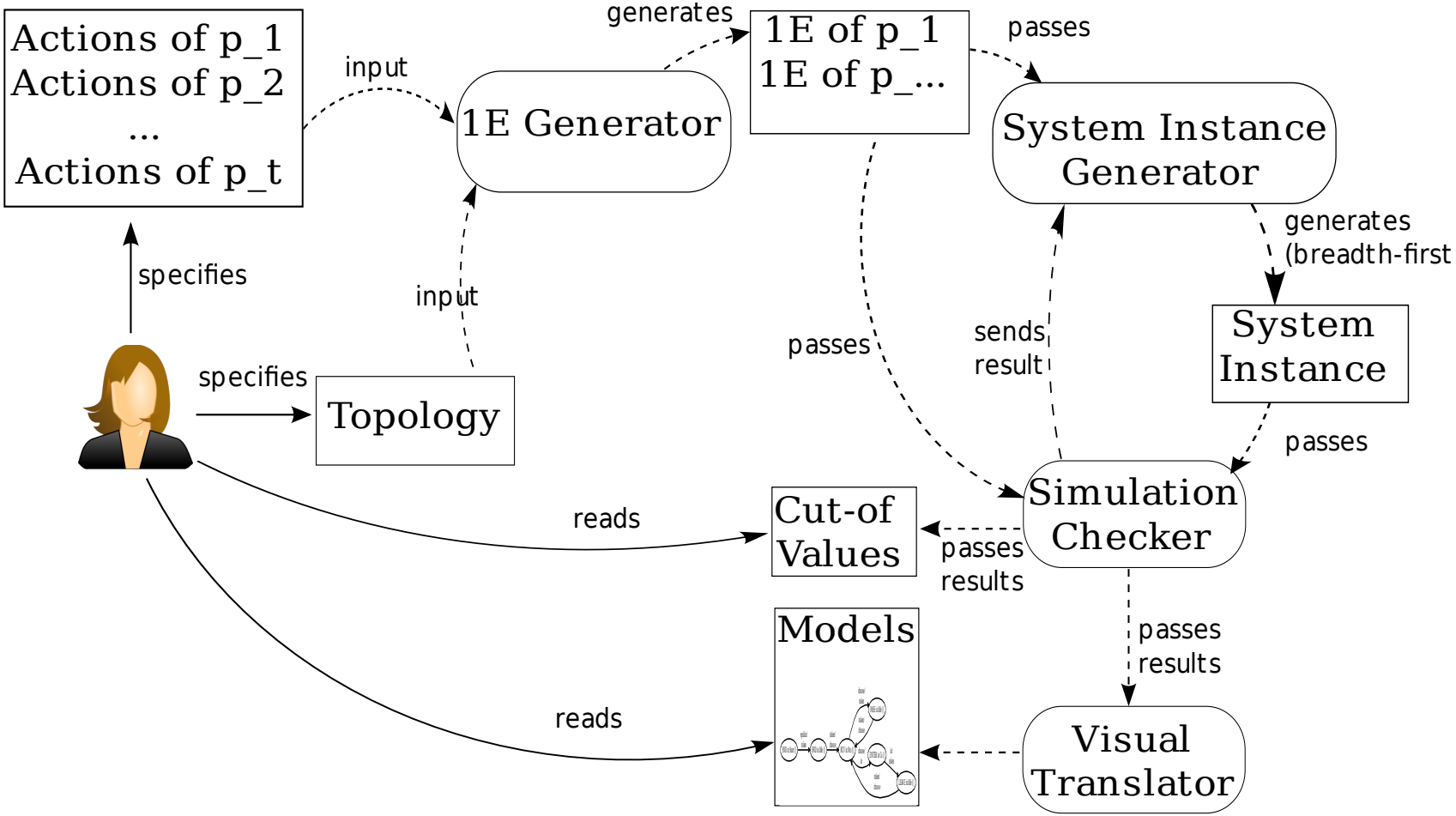
- A system with **3** philosophers of each type exhibits the maximal behaviors $1E_l$ and $1E_r$
- **cut-off for RLDP is $3_l, 3_r$**

***sys*$(3_l, 3_r)$** satisfies $\varphi \Leftrightarrow$ all larger system instances satisfy φ

Kind of Properties φ

- Properties involving one process i
- Properties involving 2 neigbor processes i, j
  - Could be of different types.

Our Tool: Golok



Comparison with Other Techniques

	Topology	Existing Work	Our Technique
		Cut-off	Cut-off
Dining Phil.	Ring	**4** *	**3**
RLDP	Ring	$\mathbf{2}_r, \mathbf{2}_l$ *	$\mathbf{3}_r, \mathbf{3}_l$
Spin Lock	Star	**3**‡	**2**
Spin Lock (Multiple Objects)	Star	X	$\mathbf{2}_t, \mathbf{2}_o$
Bounded-Buffer	Star	X	$\mathbf{2}_p, \mathbf{1}_c$

* E. A. Emerson and V. Kahlon. Model checking large-scale and parameterized resource allocation systems. In TACAS, pages 251-265, 2002.
‡ Basu, S., Ramakrishnan, C.R.: Compositional analysis for verification of parameterized systems. Theor. Comput. Sci. 2006

Conclusion

- Many systems are multi-parameterized thus comes the need to verify them.
- Reduce problem to verify small system with cut-off k.
- Current solutions are topology/parameters specific.
  - New theories required for every new system.
- **Our Solution:**
  - Automated technique for cut-off generation.
  - Topology/Parameter Independent.
- **Future Work:** Apply technique to
  - Infinite-domain data systems.

Questions?

http://www.cs.iastate.edu/~slede/golok/