Adoption Protocols for Fanout-Optimal Fault-Tolerant Termination Detection

Jonathan Lifflander, Phil Miller, Laxmikant V. Kale

{jliffl2, mille121, kale}@illinois.edu

University of Illinois Urbana-Champaign

February 26, 2013

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant

Termination Detection

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant

Termination Detection

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant

Termination Detection $\longrightarrow$ What is it?

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant

Termination Detection $\longrightarrow$ What is it?
Why is it relevant?

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant

Termination Detection $\longrightarrow$ What is it?
Why is it relevant?
Algorithm overview

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant $\longrightarrow$ Problem description

Termination Detection $\longrightarrow$ What is it?
Why is it relevant?
Algorithm overview

Adoption Protocols for

Fanout-Optimal

Fault-Tolerant → Problem description
Previous work

Termination Detection → What is it?
Why is it relevant?
Algorithm overview

Adoption Protocols for

Fanout-Optimal	$\longrightarrow$	Theoretical bounds
Fault-Tolerant	$\longrightarrow$	Problem description Previous work
Termination Detection	$\longrightarrow$	What is it? Why is it relevant? Algorithm overview

		Trio of protocols:
Adoption Protocols for	$\longrightarrow$	
Fanout-Optimal	$\longrightarrow$	Theoretical bounds
Fault-Tolerant	$\longrightarrow$	Problem description Previous work
Termination Detection	$\longrightarrow$	What is it? Why is it relevant? Algorithm overview

Adoption Protocols for	$\longrightarrow$	Trio of protocols: {INDEP, RELLAZY, RELEAGER}
Fanout-Optimal	$\longrightarrow$	Theoretical bounds
Fault-Tolerant	$\longrightarrow$	Problem description Previous work
Termination Detection	$\longrightarrow$	What is it? Why is it relevant? Algorithm overview

Adoption Protocols for $\longrightarrow$ Trio of protocols: {Indep, RelLazy, RelEager}
Probabilistic model

Fanout-Optimal $\longrightarrow$ Theoretical bounds

Fault-Tolerant $\longrightarrow$ Problem description
Previous work

Termination Detection $\longrightarrow$ What is it?
Why is it relevant?
Algorithm overview

Adoption Protocols for $\longrightarrow$ Trio of protocols:
{INDEP,RELLAZY,RELEAGER}
Probabilistic model
High survivability

Fanout-Optimal $\longrightarrow$ Theoretical bounds

Fault-Tolerant $\longrightarrow$ Problem description
Previous work

Termination Detection $\longrightarrow$ What is it?
Why is it relevant?
Algorithm overview

Adoption Protocols for	$\longrightarrow$	Trio of protocols: {INDEP,RELLAZY,RELEAGER} Probabilistic model High survivability Empirical Results
Fanout-Optimal	$\longrightarrow$	Theoretical bounds
Fault-Tolerant	$\longrightarrow$	Problem description Previous work
Termination Detection	$\longrightarrow$	What is it? Why is it relevant? Algorithm overview

Termination Detection

→ What is it?

- The “global” system state when
  - all processes are idle and
  - no messages are in flight

Termination Detection

→ What is it?

- The "global" system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.

Termination Detection

→ What is it?

- The “global” system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*

Termination Detection

→ What is it?

- The "global" system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*
  - A process starts passive (except for the root)

Termination Detection

→ What is it?

- The “global” system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*
  - A process starts passive (except for the root)
  - A process only becomes active when it receives a message

Termination Detection

→ What is it?

- The “global” system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*
  - A process starts passive (except for the root)
  - A process only becomes active when it receives a message
  - A process can only send messages in the active state

Termination Detection

→ What is it?

- The "global" system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*
  - A process starts passive (except for the root)
  - A process only becomes active when it receives a message
  - A process can only send messages in the active state
  - A process transitions back to passive after processing a message

Termination Detection

→ What is it?

- The "global" system state when
  - all processes are idle and
  - no messages are in flight
- Fundamental assumptions
  - A process is any schedulable entity: thread, object, etc.
  - A process is either *active* or *passive*
  - A process starts passive (except for the root)
  - A process only becomes active when it receives a message
  - A process can only send messages in the active state
  - A process transitions back to passive after processing a message
  - idle $\equiv$ passive

Termination Detection

→ Why is it relevant?

- As parallel computations become more dynamic, detecting termination is non-trivial

Termination Detection

→ Why is it relevant?

- As parallel computations become more dynamic, detecting termination is non-trivial
  - Adaptive mesh refinement*

* Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, Paul Ricker. *Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement*. In Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '12).

Termination Detection

→ Why is it relevant?

- As parallel computations become more dynamic, detecting termination is non-trivial
  - Adaptive mesh refinement*
  - Dynamic data exchanges (e.g. SPMV)

* Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, Paul Ricker. *Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement*. In Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '12).

Termination Detection

→ Why is it relevant?

- As parallel computations become more dynamic, detecting termination is non-trivial
  - Adaptive mesh refinement[*]
  - Dynamic data exchanges (e.g. SPMV)
  - Distributed-memory work stealing – task scheduling[†]

[*] Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, Paul Ricker. *Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement*. In Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '12).

[†] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. *Scalable work stealing.* In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09).

Termination Detection

→ Why is it relevant?

- As parallel computations become more dynamic, detecting termination is non-trivial
  - Adaptive mesh refinement[*]
  - Dynamic data exchanges (e.g. SPMV)
  - Distributed-memory work stealing – task scheduling[†]
  - Runtimes with message-driven execution[‡]

[*] Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, Paul Ricker. *Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement*. In Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '12).

[†] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. *Scalable work stealing.* In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09).

[‡] Jeremiah Willcock, Torsten Hoefler, Nicholas Edmonds, Andrew Lumsdaine. *Active pebbles: parallel programming for data-driven applications*. Proceedings of the international conference on Supercomputing. ICS'11.

Termination Detection

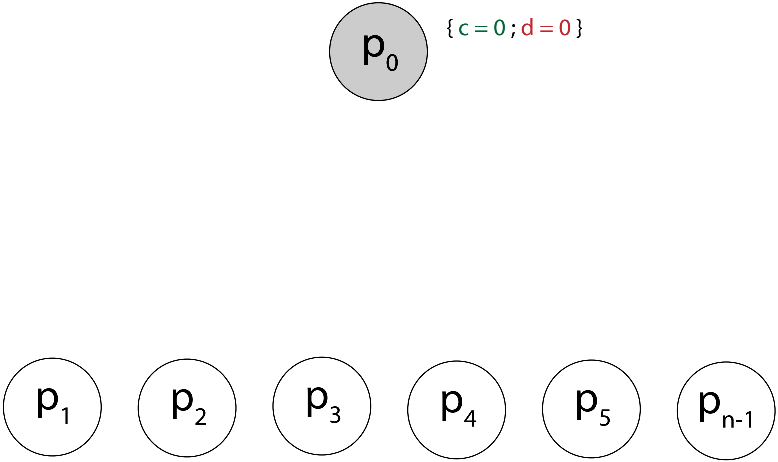
→ Algorithms

- Many different variants (with different tradeoffs)
  - Parental responsibility[§]
  - Wave-based
  - Credit-recovery

[§] Edsger W. Dijkstra, C.S. Scholten. *Termination detection for diffusing computations.* Information Processing Letters. 1980.

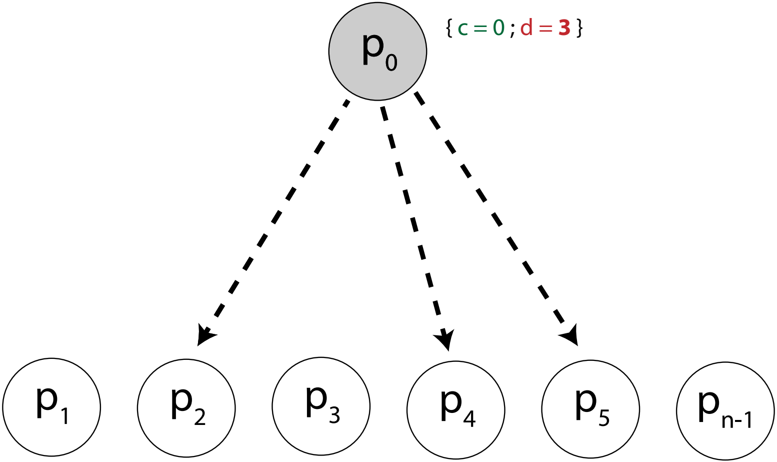
Termination Detection

→ Algorithm Overview : Parental Responsibility



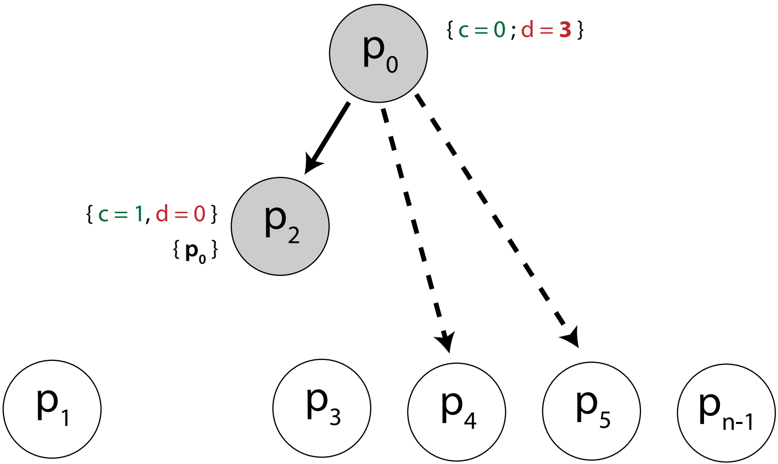
Termination Detection

→ Algorithm Overview : Parental Responsibility



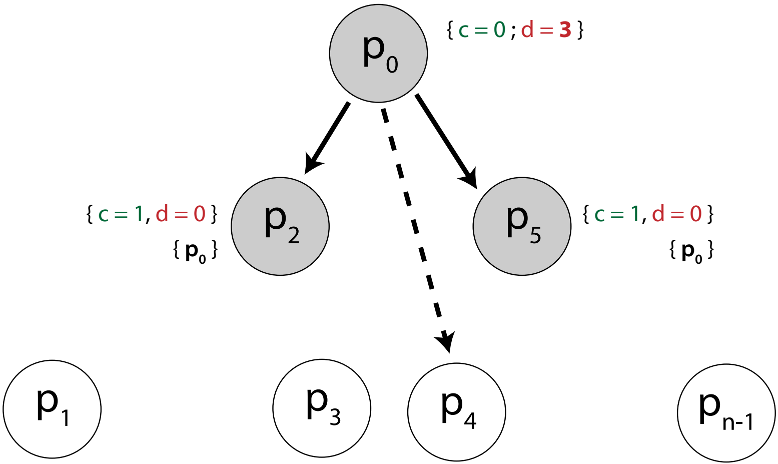
Termination Detection

→ Algorithm Overview : Parental Responsibility



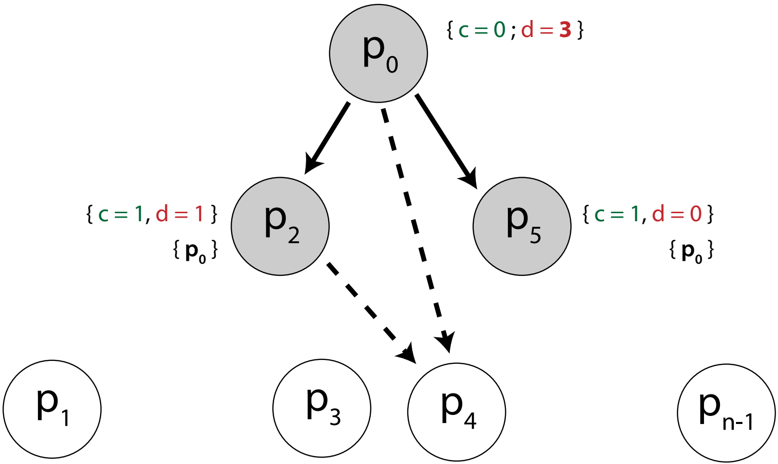
Termination Detection

→ Algorithm Overview : Parental Responsibility



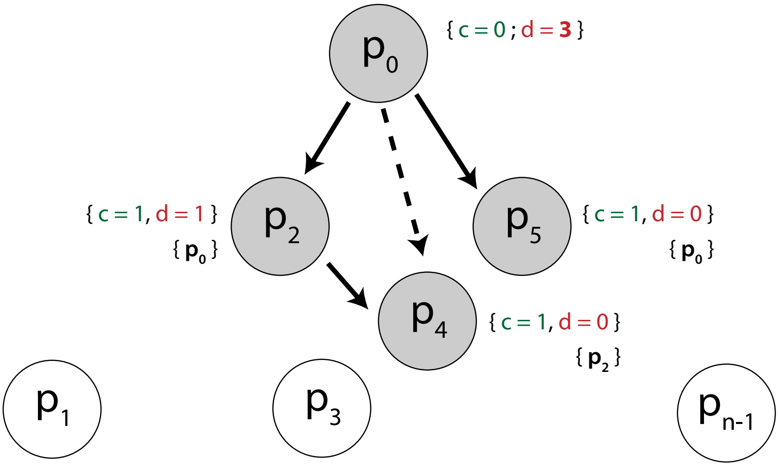
Termination Detection

→ Algorithm Overview : Parental Responsibility



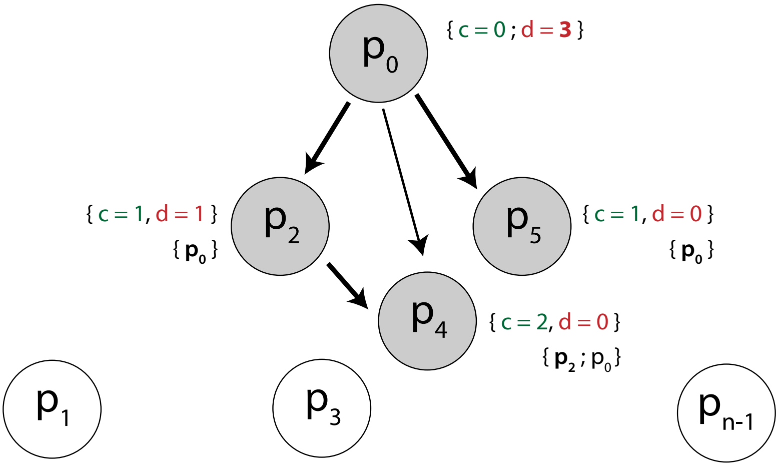
Termination Detection

→ Algorithm Overview : Parental Responsibility



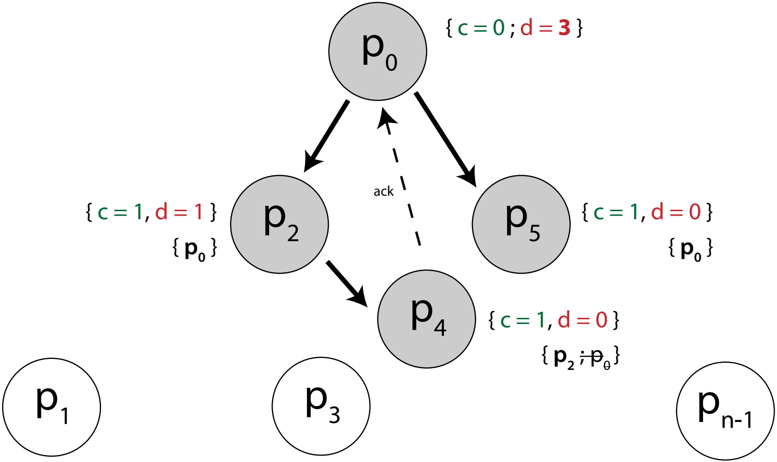
Termination Detection

→ Algorithm Overview : Parental Responsibility



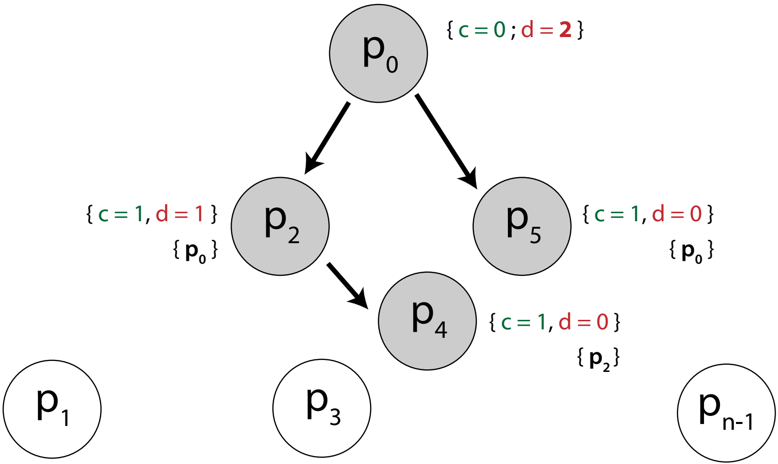
Termination Detection

→ Algorithm Overview : Parental Responsibility



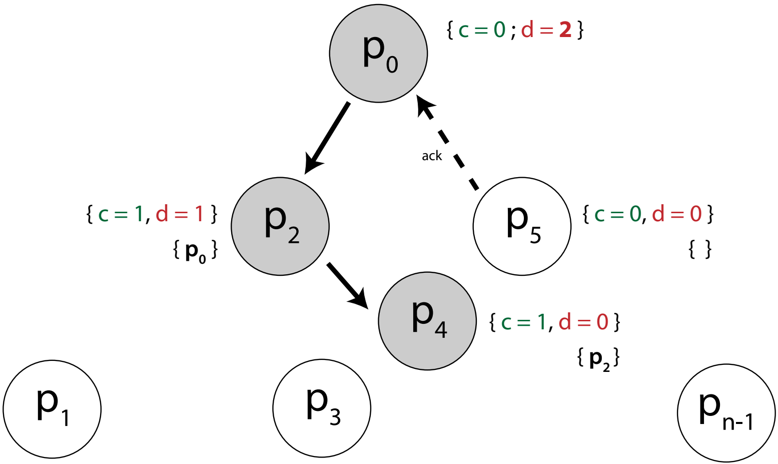
Termination Detection

→ Algorithm Overview : Parental Responsibility



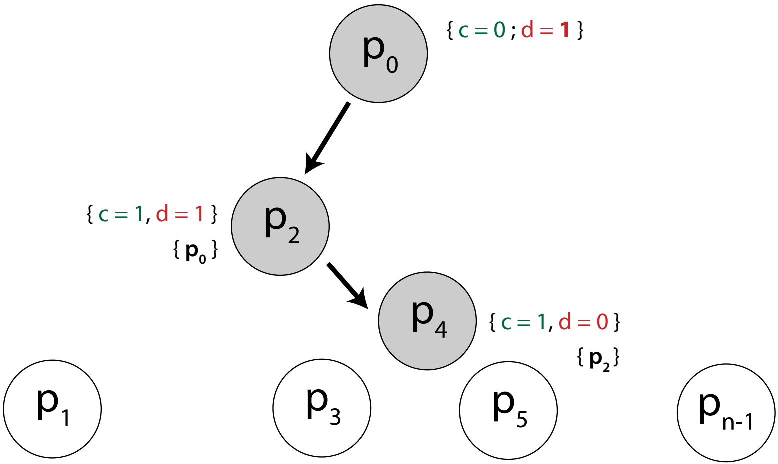
Termination Detection

→ Algorithm Overview : Parental Responsibility



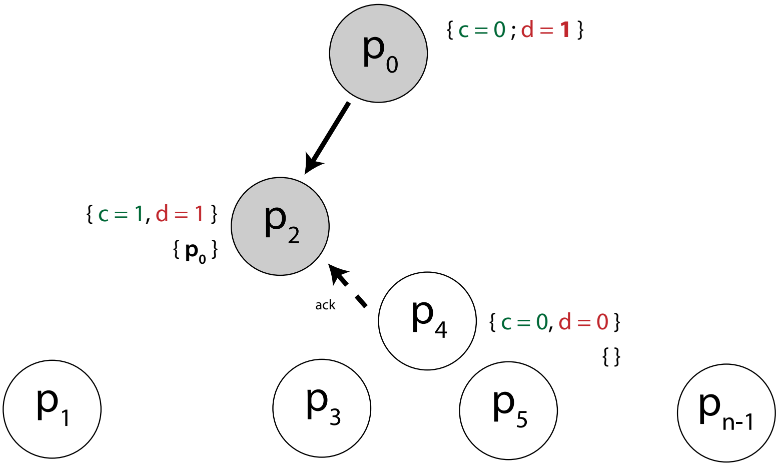
Termination Detection

→ Algorithm Overview : Parental Responsibility



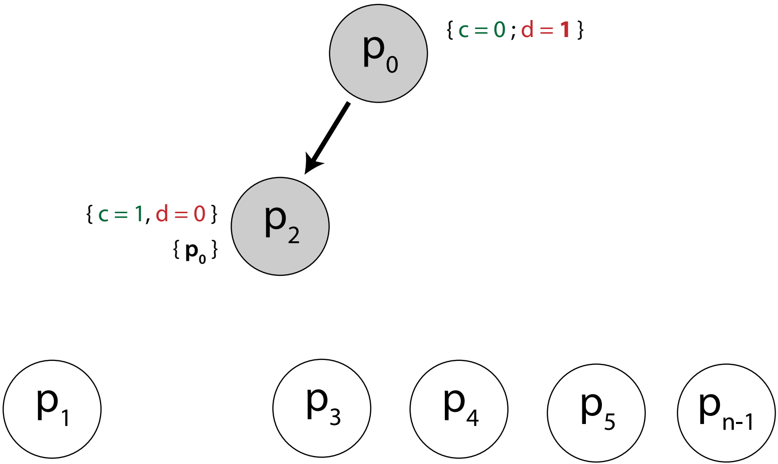
Termination Detection

→ Algorithm Overview : Parental Responsibility



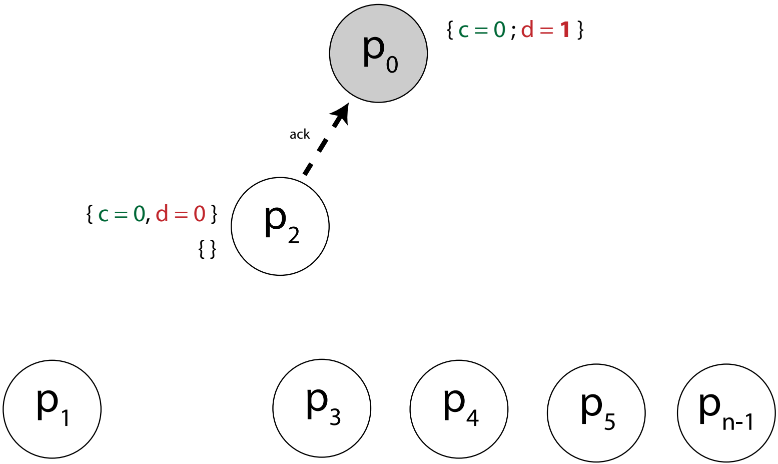
Termination Detection

→ Algorithm Overview : Parental Responsibility



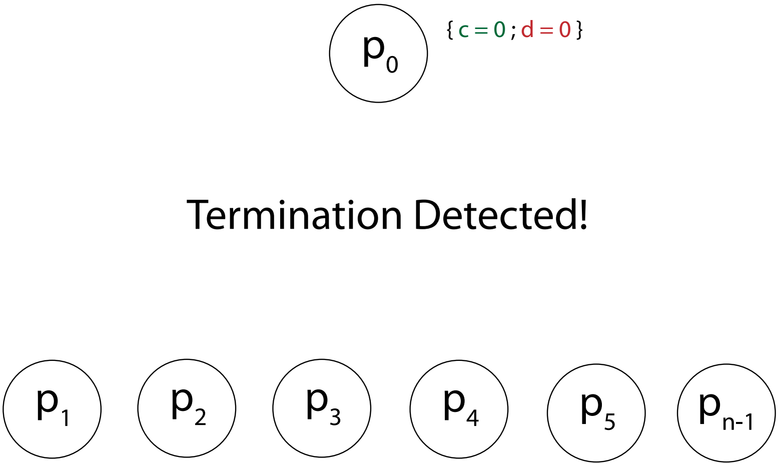
Termination Detection

→ Algorithm Overview : Parental Responsibility



Termination Detection

→ Algorithm Overview : Parental Responsibility



git://charm.cs.illinois.edu/terminator.git

`git://charm.cs.illinois.edu/terminator.git`

General C++ and Java library

`git://charm.cs.illinois.edu/terminator.git`

Implemented in three parallel runtime systems

`git://charm.cs.illinois.edu/terminator.git`

Being made fault-tolerant based on this work

Fault-Tolerant Termination Detection

→ Problem Description

- Approaches to fault-tolerance
  - General runtime-system support: checkpointing, message-logging, etc.

Fault-Tolerant Termination Detection

→ Problem Description

- Approaches to fault-tolerance
  - General runtime-system support: checkpointing, message-logging, etc.
  - Algorithm-specific (checksum-based approaches for math libraries)

Fault-Tolerant Termination Detection

→ Problem Description

- Approaches to fault-tolerance
  - General runtime-system support: checkpointing, message-logging, etc.
  - Algorithm-specific (checksum-based approaches for math libraries)
  - Component-specific: runtime system is composed of a set of self-healing components that all handle faults in their own optimized way — so-called *scale invariance*[¶]

[¶] Al Geist and Christian Engelmann. *Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors*. 2002.

Fault-Tolerant Termination Detection

$\rightarrow$ Component-specific

- Assume ecosystem is fault tolerant
  - Application can recover from faults
  - Other runtime system components are fault-tolerant
  - Termination can be handled as a modular component

Fault-Tolerant Termination Detection

→ Previous work

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*[‖]

[‖] Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection*. Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ Previous work

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*[‖]
  - Recovers from up to n-1 faults in the system!

[‖] Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection*. Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ Previous work

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*[‖]
  - Recovers from up to n-1 faults in the system!
  - But not practical for HPC

[‖] Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection*. Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ Previous work

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*‖
  - Recovers from up to n-1 faults in the system!
  - But not practical for HPC
  - Serializes recovery through the root (one process) of the computation

‖Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection*. Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ **Previous work**

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*[‖]
  - Recovers from up to n-1 faults in the system!
  - But not practical for HPC
  - Serializes recovery through the root (one process) of the computation
  - Requires all processes (even unengaged) to communicate (obviating many beneficial properties of parental responsibility algorithms)

[‖] Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection*. Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ Previous work

- Distributed computing
  - *An (n-1)-resilient algorithm for distributed termination detection.*[‖]
  - Recovers from up to n-1 faults in the system!
  - But not practical for HPC
  - Serializes recovery through the root (one process) of the computation
  - Requires all processes (even unengaged) to communicate (obviating many beneficial properties of parental responsibility algorithms)
  - Has low overhead on forward execution

[‖] Ten-Hwang Lai, Li-Fen Wu. *An (n-1)-resilient algorithm for distributed termination detection.* Parallel and Distributed Systems, IEEE Transactions. Vol.6, No.1, pp.63-78, Jan 1995.

Fault-Tolerant Termination Detection

→ Goals for HPC

- Low overhead during forward execution
- Expect to encounter faults that only impact a small subset of the processes
- Build a scalable algorithm for recovery

Termination Detection

→ Optimality

- Parental responsibility algorithms
  - *Message-optimal*: in the worst case, they send the lower-bound on signal count**
  - Where the lower bound is $\mathcal{O}(m)$, and m is the total number of application messages

**K. Mani Chandy and Jayadev Misra. *How processes learn*. In Proceedings of the forth annual ACM symposium on Principles of Distributed Computing (PODC '85).

Termination Detection

→ Optimality

- *Fanout-optimality*
  - The *fanout* or f for a given process is the number of communication partners it has

Termination Detection

→ Optimality

- *Fanout-optimality*
  - The *fanout* or f for a given process is the number of communication partners it has
  - Because it follows the communication graph of the application, it will be as scalable as the application

Termination Detection

→ Optimality

- *Fanout-optimality*
  - The *fanout* or f for a given process is the number of communication partners it has
  - Because it follows the communication graph of the application, it will be as scalable as the application
  - For parental-responsibility algorithms, this is a dynamic property

Termination Detection

$\rightarrow$ Optimality

- *Fanout-optimality*
  - The *fanout* or f for a given process is the number of communication partners it has
  - Because it follows the communication graph of the application, it will be as scalable as the application
  - For parental-responsibility algorithms, this is a dynamic property
  - Optimality: in the worst case, the algorithm sends $\mathcal{O}(f)$ signals

Termination Detection

→ Optimality

- *Fanout-optimality*
  - The *fanout* or f for a given process is the number of communication partners it has
  - Because it follows the communication graph of the application, it will be as scalable as the application
  - For parental-responsibility algorithms, this is a dynamic property
  - Optimality: in the worst case, the algorithm sends $\mathcal{O}(f)$ signals
  - The fault-tolerance algorithms we present are fanout-optimal

Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover

Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover
  - External system (or runtime system) provides failure notification

Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover
  - External system (or runtime system) provides failure notification
  - No byzantine failures: failed processes do not behave maliciously

Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover
  - External system (or runtime system) provides failure notification
  - No byzantine failures: failed processes do not behave maliciously
- Specific

Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover
  - External system (or runtime system) provides failure notification
  - No byzantine failures: failed processes do not behave maliciously
- Specific
  - *Network send fence*: messages are "on-the-wire"

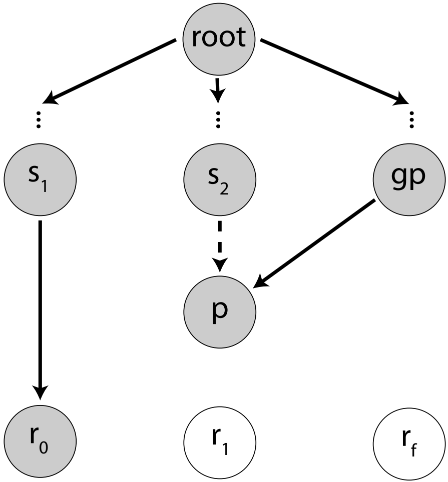
Termination Detection

→ Fault-tolerance assumptions

- General
  - *Fail-stop* model: failed processes do not recover
  - External system (or runtime system) provides failure notification
  - No byzantine failures: failed processes do not behave maliciously
- Specific
  - *Network send fence*: messages are "on-the-wire"
  - *Fail-flush*: a system notification indicating that in-flight messages from a failed process have all been received

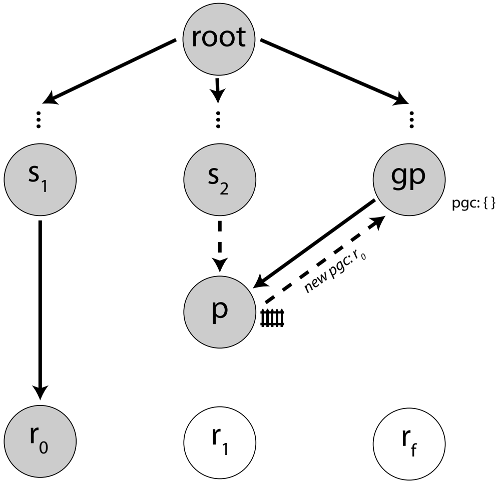
Termination Detection

→ The INDEP fault-tolerance protocol



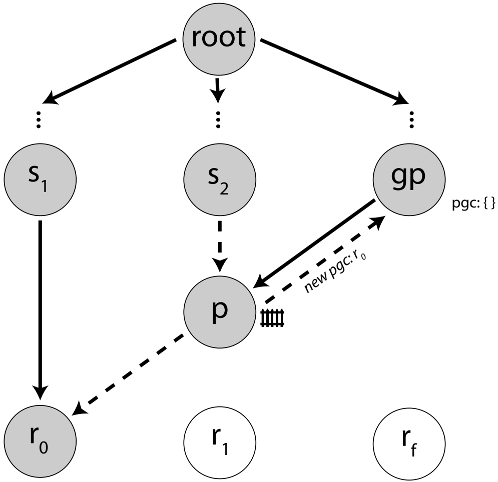
Termination Detection

→ The INDEP fault-tolerance protocol



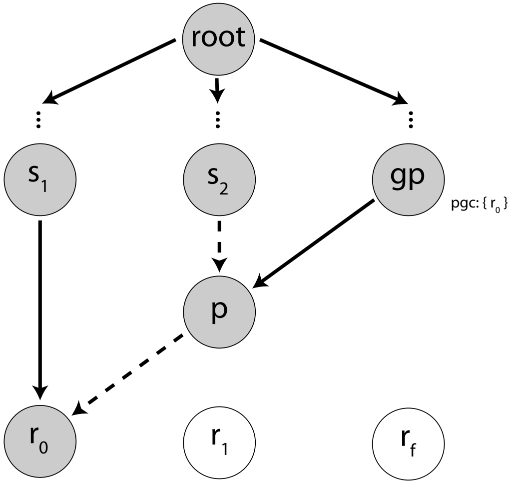
Termination Detection

→ The INDEP fault-tolerance protocol



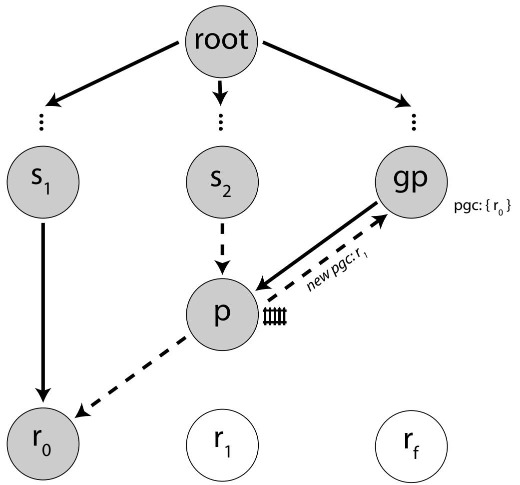
Termination Detection

→ The INDEP fault-tolerance protocol



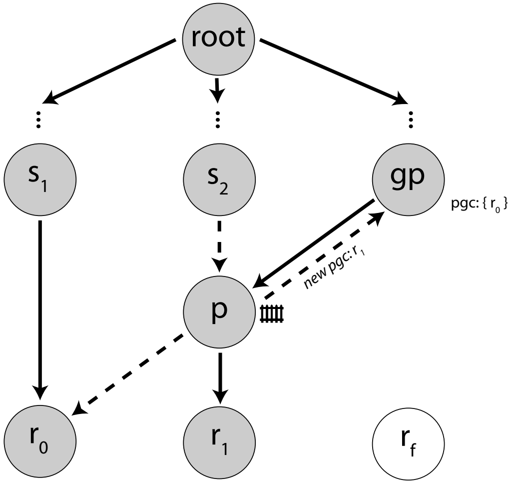
Termination Detection

→ The INDEP fault-tolerance protocol



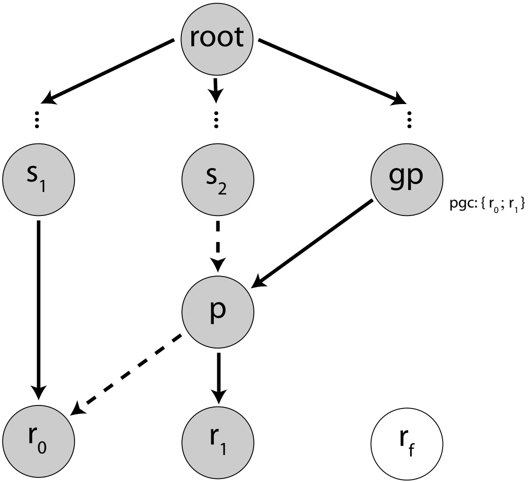
Termination Detection

→ The INDEP fault-tolerance protocol



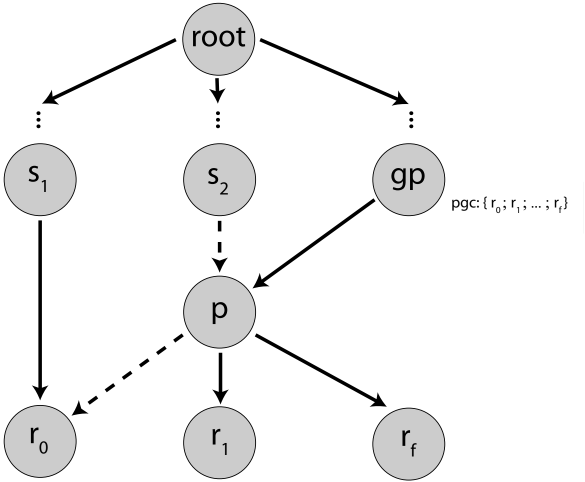
Termination Detection

→ The INDEP fault-tolerance protocol



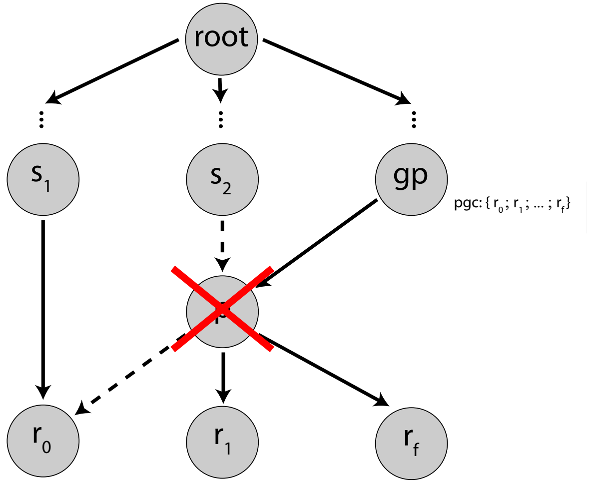
Termination Detection

→ The INDEP fault-tolerance protocol



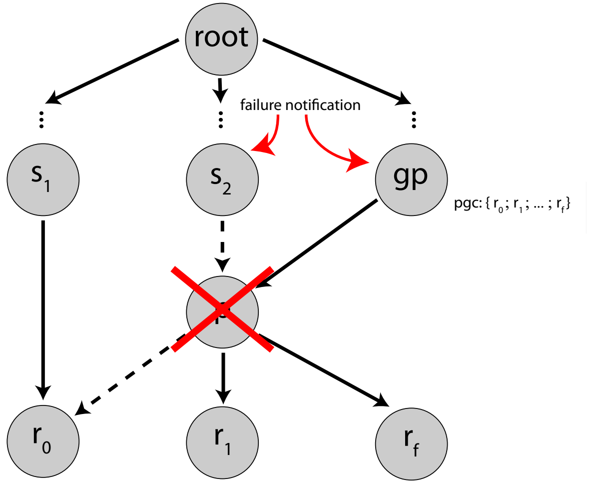
Termination Detection

→ The INDEP fault-tolerance protocol



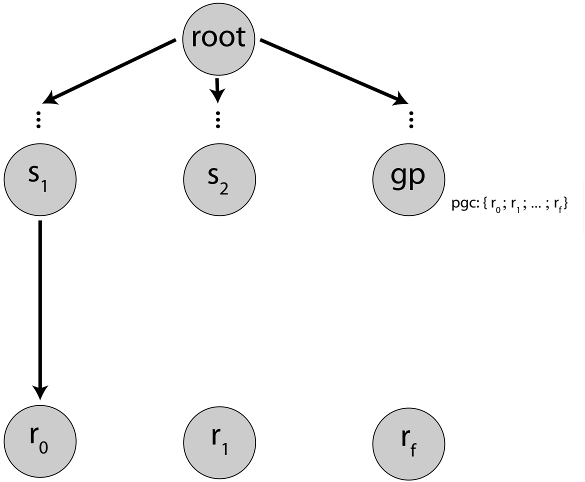
Termination Detection

→ The INDEP fault-tolerance protocol



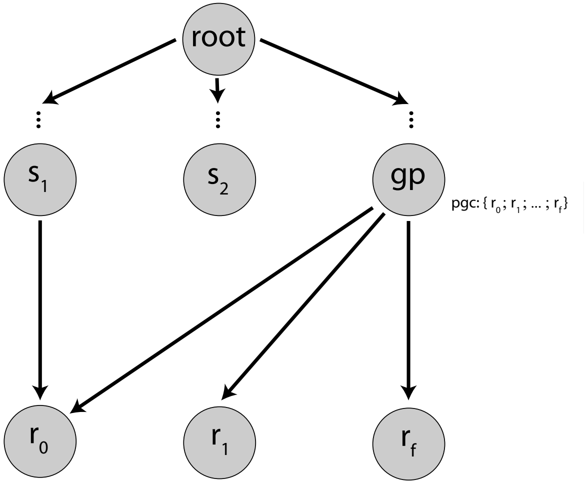
Termination Detection

→ **The INDEP fault-tolerance protocol**



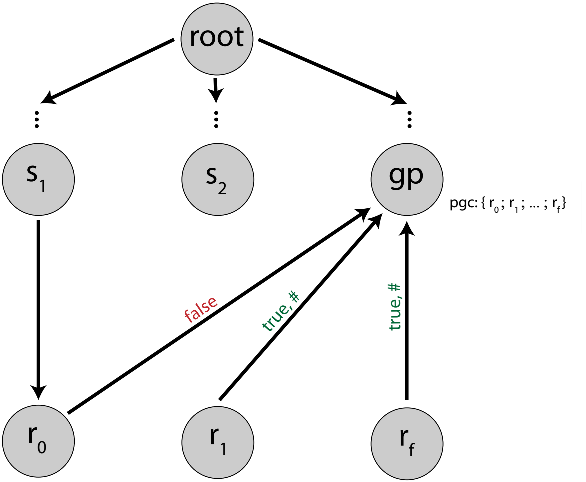
Termination Detection

→ The INDEP fault-tolerance protocol



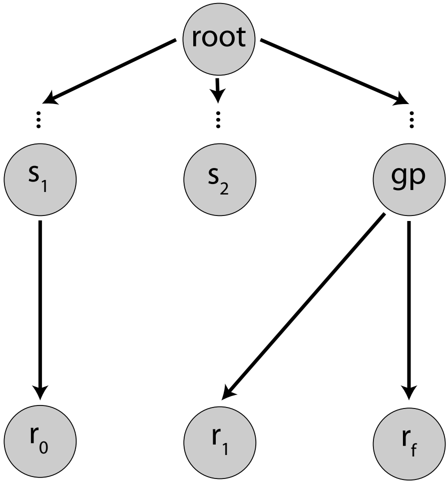
Termination Detection

→ The INDEP fault-tolerance protocol



Termination Detection

→ The INDEP fault-tolerance protocol



Termination Detection

→ The INDEP fault-tolerance protocol

- Tolerates all single-process failures

Termination Detection

→ The INDEP fault-tolerance protocol

- Tolerates all single-process failures
- What multi-process failures does it not tolerate?

Termination Detection

→ The INDEP fault-tolerance protocol

- Tolerates all single-process failures
- What multi-process failures does it not tolerate?
  - INDEP will tolerate all failures except for parent-child pairs...

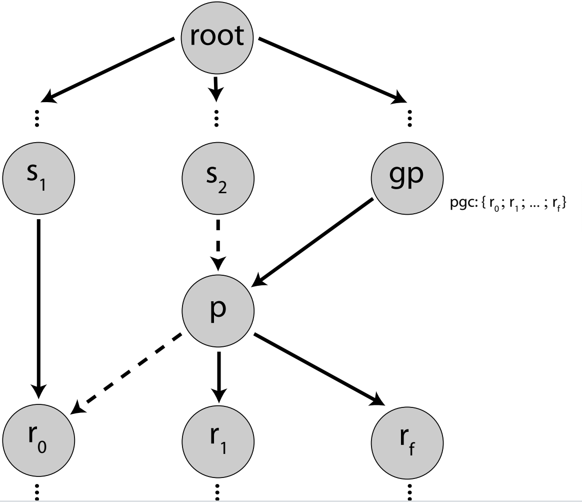
Termination Detection

→ The INDEP fault-tolerance protocol

- Tolerates all single-process failures
- What multi-process failures does it not tolerate?
  - INDEP will tolerate all failures except for parent-child pairs...
  - But, INDEP cannot detect this case, so it has to fail conservatively if the failure set has communicating pairs in it

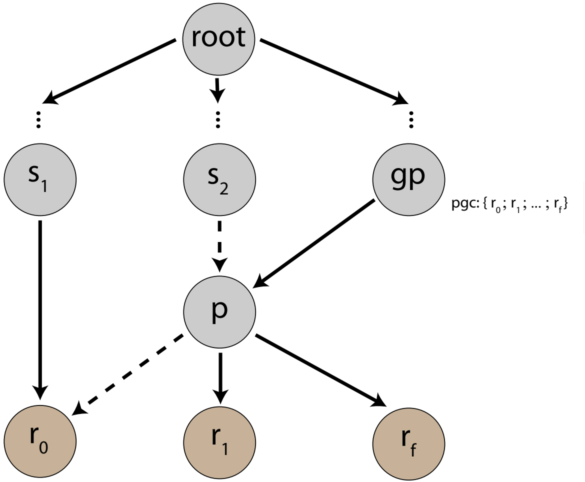
Termination Detection

→ The INDEP fault-tolerance protocol



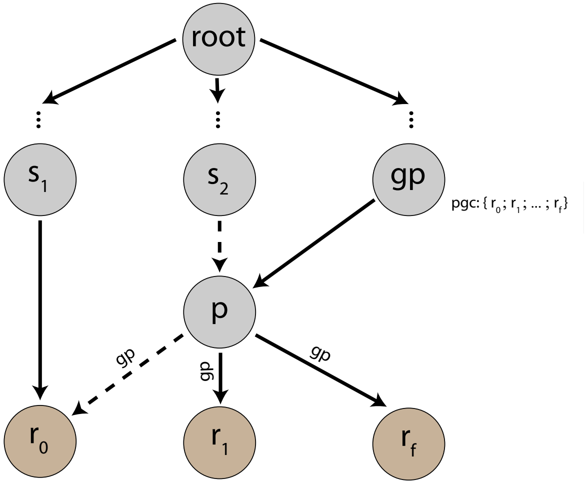
Termination Detection

→ **The** RELLAZY **and** RELEAGER **protocols**



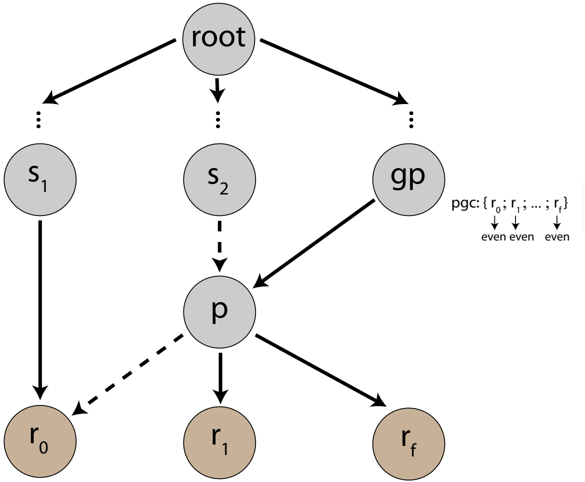
Termination Detection

→ **The** RELLAZY **and** RELEAGER **protocols**



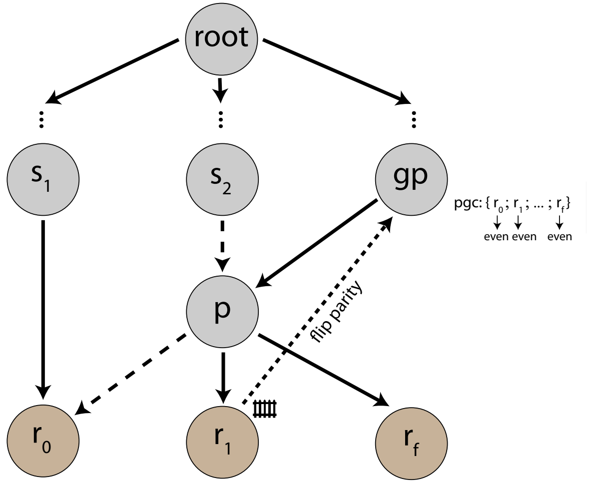
Termination Detection

→ The RelLazy and RelEager protocols



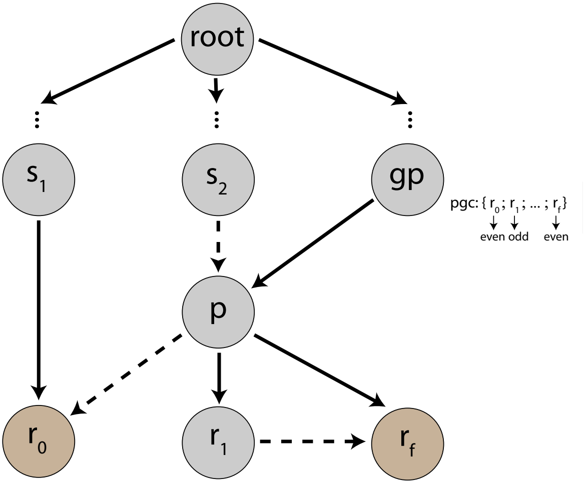
Termination Detection

→ The RelLazy and RelEager protocols



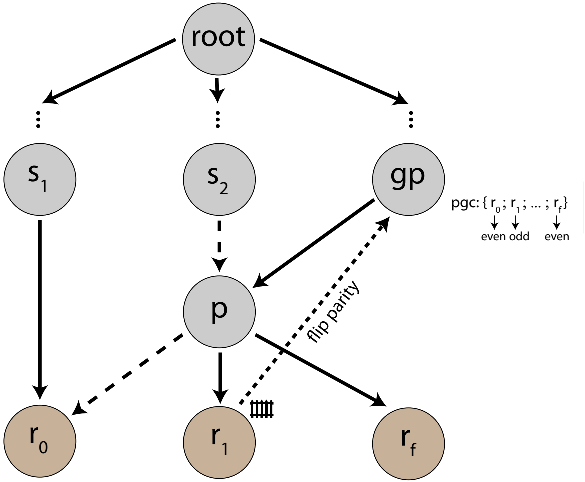
Termination Detection

→ **The RelLazy and RelEager protocols**



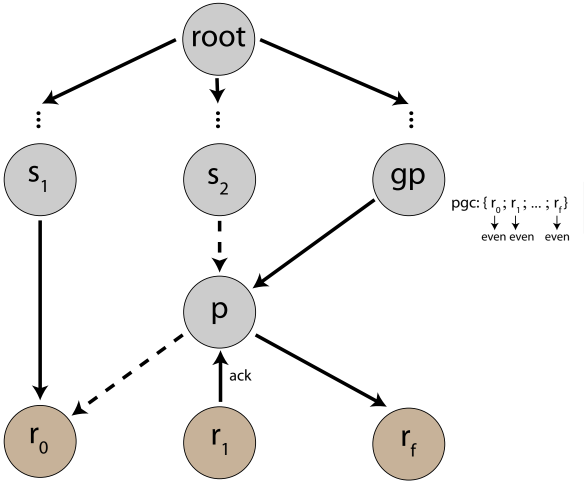
Termination Detection

→ The RelLazy and RelEager protocols



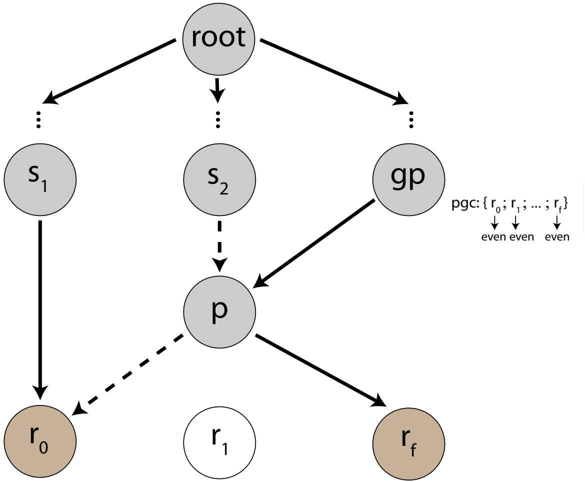
Termination Detection

→ The RELLAZY and RELEAGER protocols



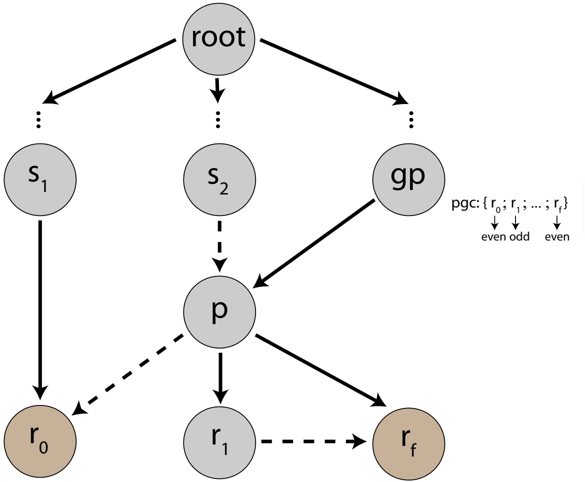
Termination Detection

→ **The RelLazy and RelEager protocols**



Termination Detection

→ The RelLazy and RelEager protocols



Termination Detection

→ Probability Model and Survivability

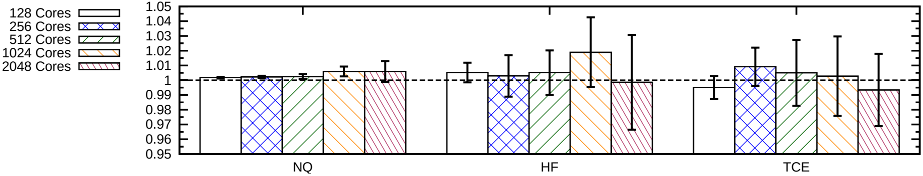
Nodes Failed	Fault (%)
1	92.30
2	3.672
3	0.942
4	0.753
5	0.565
6	0.094
7	0.094
8	0.377
9	0.094
10	0.188
11	0.188
15	0.282
18	0.094
26	0.094
86	0.094
126	0.094

Protocol		Survivability* (%)
INDEP	$f = 2$	99.32
	$f = 8$	98.63
	$f = 32$	97.47
	$f = 512$	93.21
REL*		99.50

*Assuming a 1024-node job

Termination Detection

→ Empirical Results



- Ratio of execution time without FT compared to using the REL* protocols
- Sample size of 24, using a Student's t-test, error bars represent standard error at 99% confidence
- Using distributed-memory work stealing — NQueens (NQ), HF (Hartree-Fock), TCE (Tensor Contraction Expressions)

Termination Detected!

`git://charm.cs.illinois.edu/terminator.git`