Life, Death, and the Critical Transition: Finding Liveness Bugs in System Code

Charles Killian, James W. Anderson, Ranjit Jhala, and Amin Vahdat

Presented by Nick Sumner

25 March 2008

Background

We already know the story:

- Process cooperation difficult
- Debugging even harder
  - Finding and reproducing bugs is painful

Background

How to attack: Model Checking

Shortcomings:

- Scalability
- *Safety* v. *Liveness* expressiveness
- Must 'know cause' of bug to find it

Can *heuristically* detect liveness 'violations'

Background

Life

- Future progress is possible

Death

- Future progress is impossible. Liveness is violated.

Critical Transition

- A single step that disallows all future progress

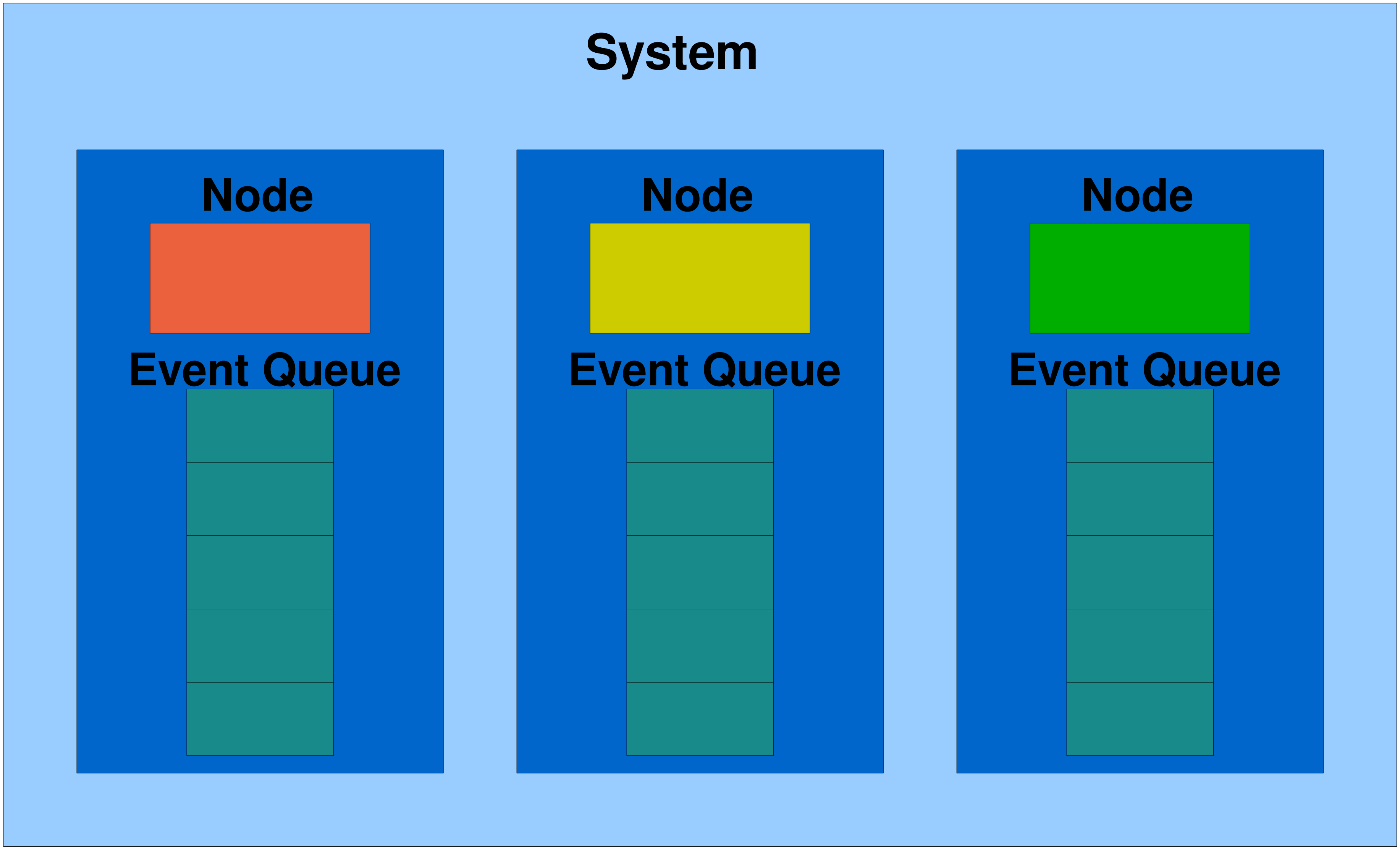
Apply random execution to find ↑

Underlying Model

- Combine all network nodes & simulate together
- **State**: (values $\times$ variables)
- **Transition**: (event $\times$ state) $\rightarrow$ state
- **Program**: (variables $\times$ $state_0$ $\times$ transitions)
- **Execution**: $\forall i=0,\ldots,\infty : state_i$

  $\wedge\ transition_k=(event,state_k) \rightarrow state_{k+1}$

Underlying Model



Each step, select one event & transition

Underlying Model

- Given predicate P over state S:
  - S ≅ Live, Dead, or Transient w.r.p. P

- **Transient**- state does not satisfy,

  but it *could* eventually

Execution violates P ⇒ ∃ state suffix w/o live states.

⇒No recovery possible

Why Not Safety Properties?

System	Name	Property	
Pastry	AllNodes	*Eventually*	$\forall n \in \textbf{nodes} : n.(successor)^* \equiv \textbf{nodes}$ Test that all nodes are reached by following successor pointers from each node.
	SizeMatch	*Always*	$\forall n \in \textbf{nodes} : n.myright.size() + n.myleft.size() = n.myleafset.size()$ Test the sanity of the leafset size compared to left and right set sizes.
Chord	AllNodes	*Eventually*	$\forall n \in \textbf{nodes} : n.(successor)^* \equiv \textbf{nodes}$ Test that all nodes are reached by following successor pointers from each node.
	SuccPred	*Always*	$\forall n \in \textbf{nodes} : \{n.predecessor = n.me \iff n.successor = n.me\}$ Test that a node's predecessor is itself if and only if its successor is itself.
RandTree	OneRoot	*Eventually*	**for** *exactly* $1\ n \in \textbf{nodes} : n.isRoot$ Test that exactly one node believes itself to be the root node.
	Timers	*Always*	$\forall n \in \textbf{nodes} : \{(n.state = init) \| (n.recovery.nextScheduled() \neq 0)\}$ Test that either the node state is $init$, or the recovery timer is scheduled.
MaceTransport	AllAcked	*Eventually*	$\forall n \in \textbf{nodes} : n.inflightSize() = 0$ Test that no messages are in-flight (i.e., not acknowledged).
			No corresponding safety property identified.

Simplicity, Expressiveness, Predictability

Process

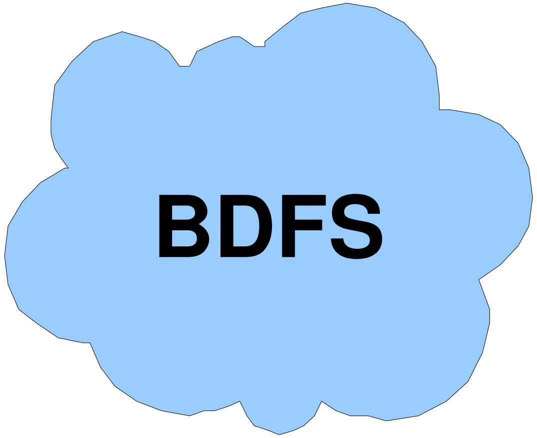
1)Bounded DFS

2)Bounded Random Walks

3)Critical Transition Isolation

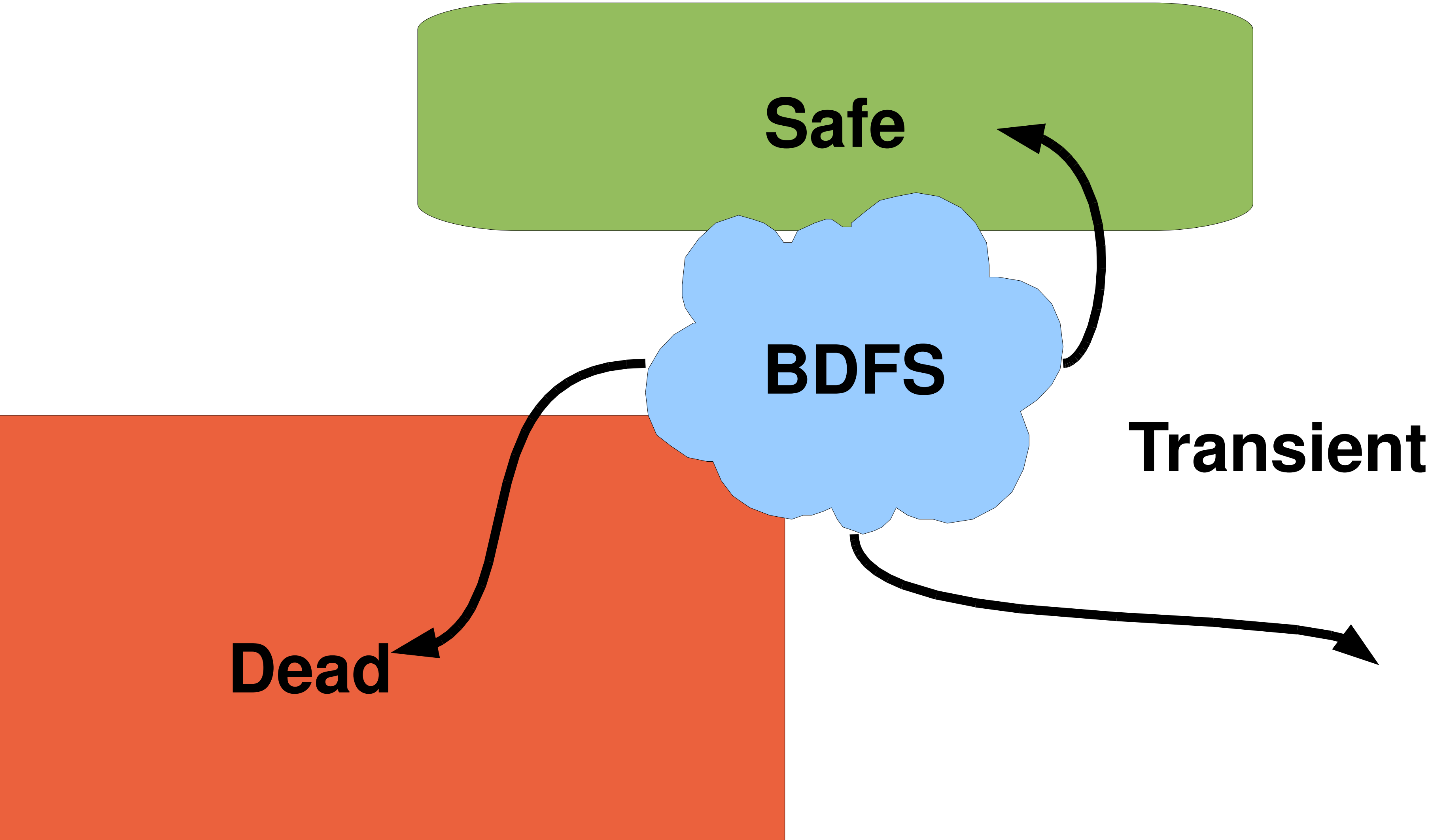
Process

- Exhaustive exploration



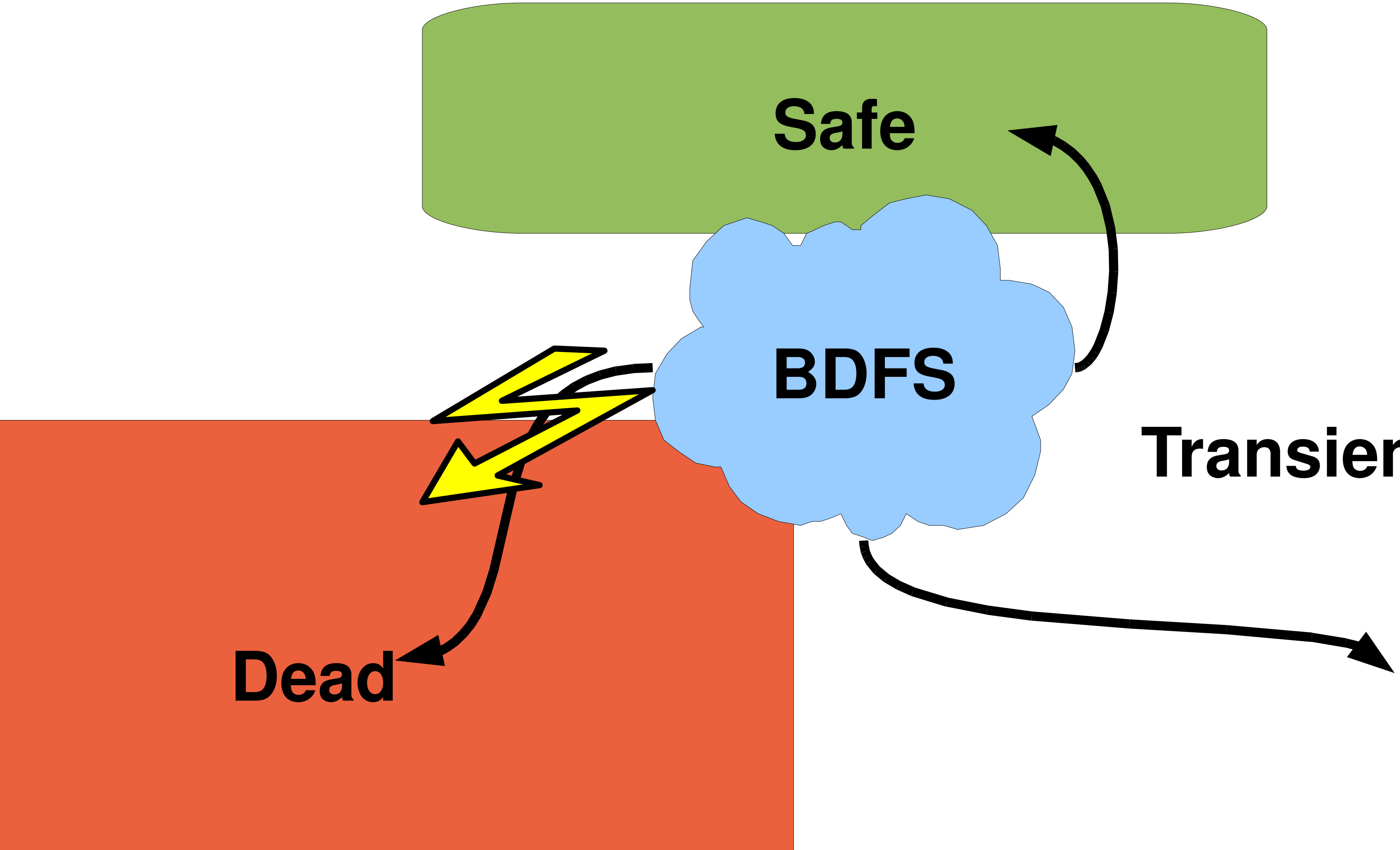
Process

- Bounded Random Walks



Process

- Critical Section Isolation



Bounded Walks

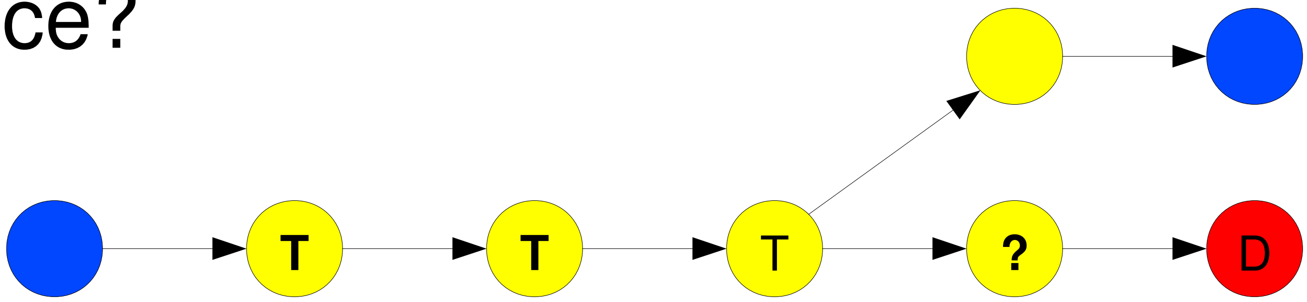
- **BDFS**- Find all valid permutations of transition sequence length **`depth`**
- **Bounded Random Walk**
  - Safety violations terminate
  - If beyond threshold and live, disregard
  - If walk through **`max`** steps, flag as possible violation

Critical Transition Isolation

Flagged executions either:

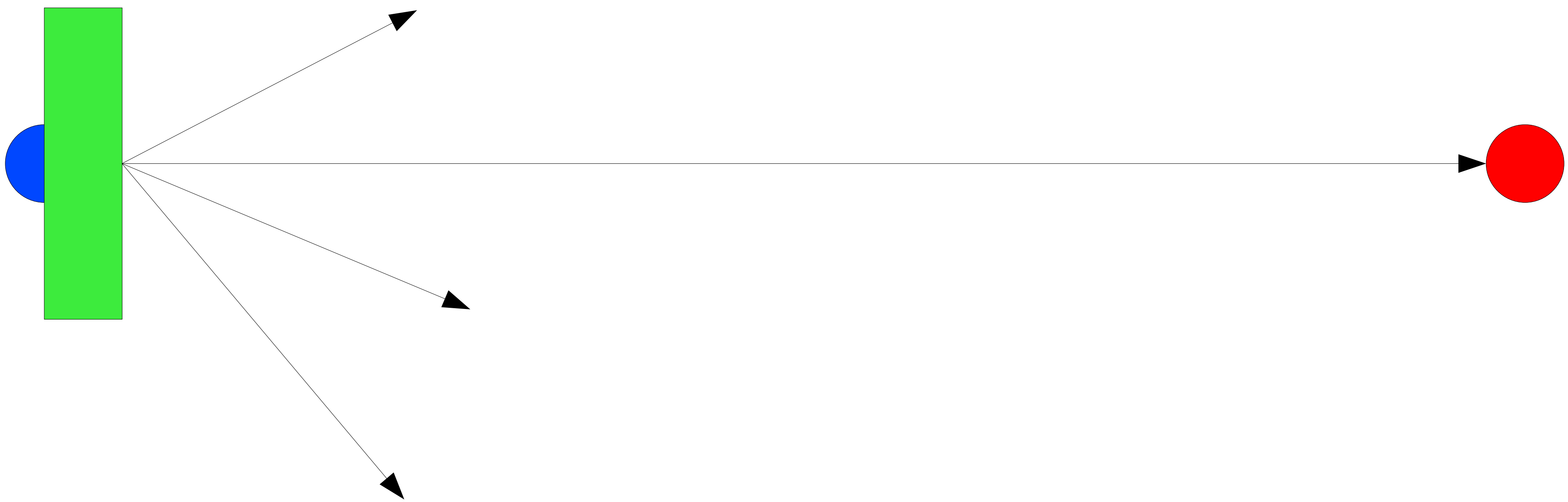
- reached a 'dead state' and must be fixed
- are still transitional and can be examined manually or with high search depth.

Difference?



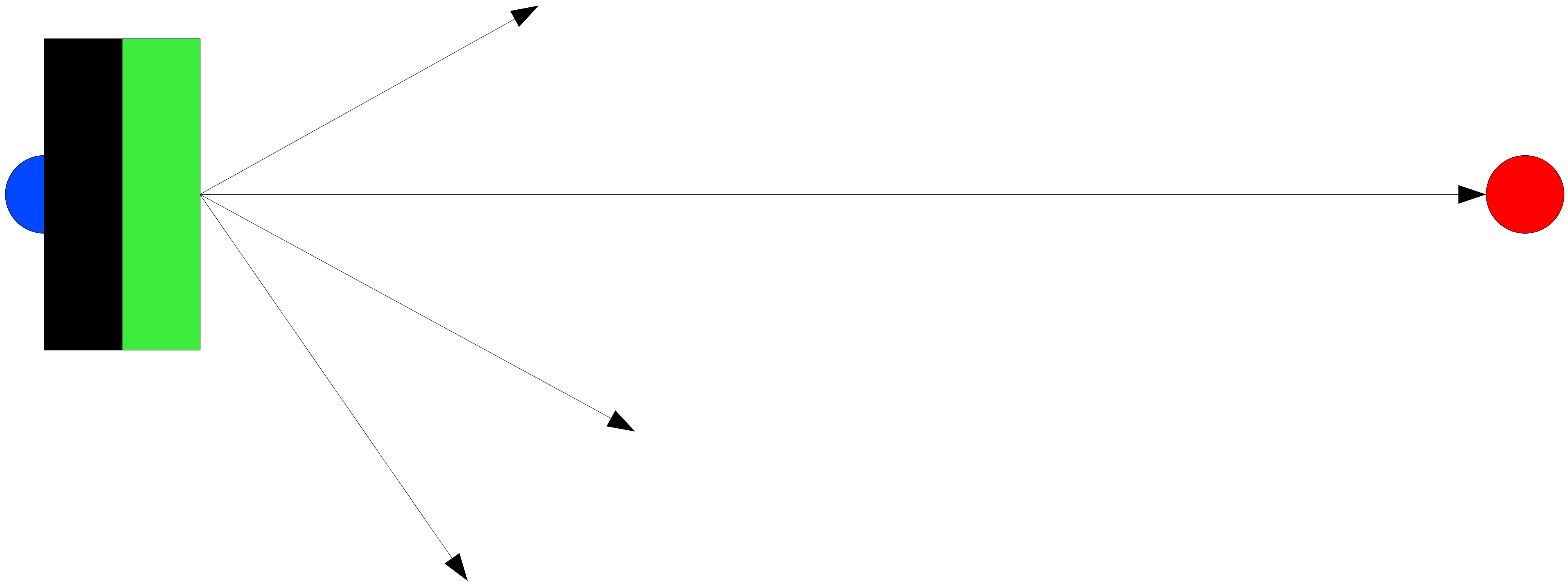
Critical Transition Isolation

Run k random walks from search edge

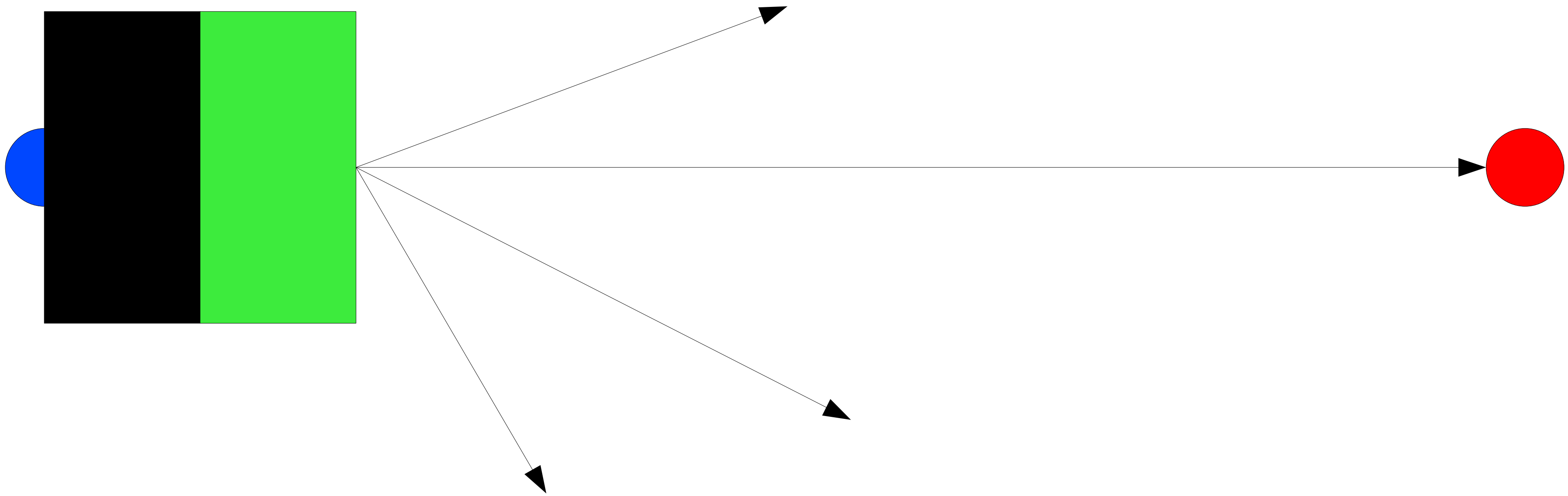


Critical Transition Isolation

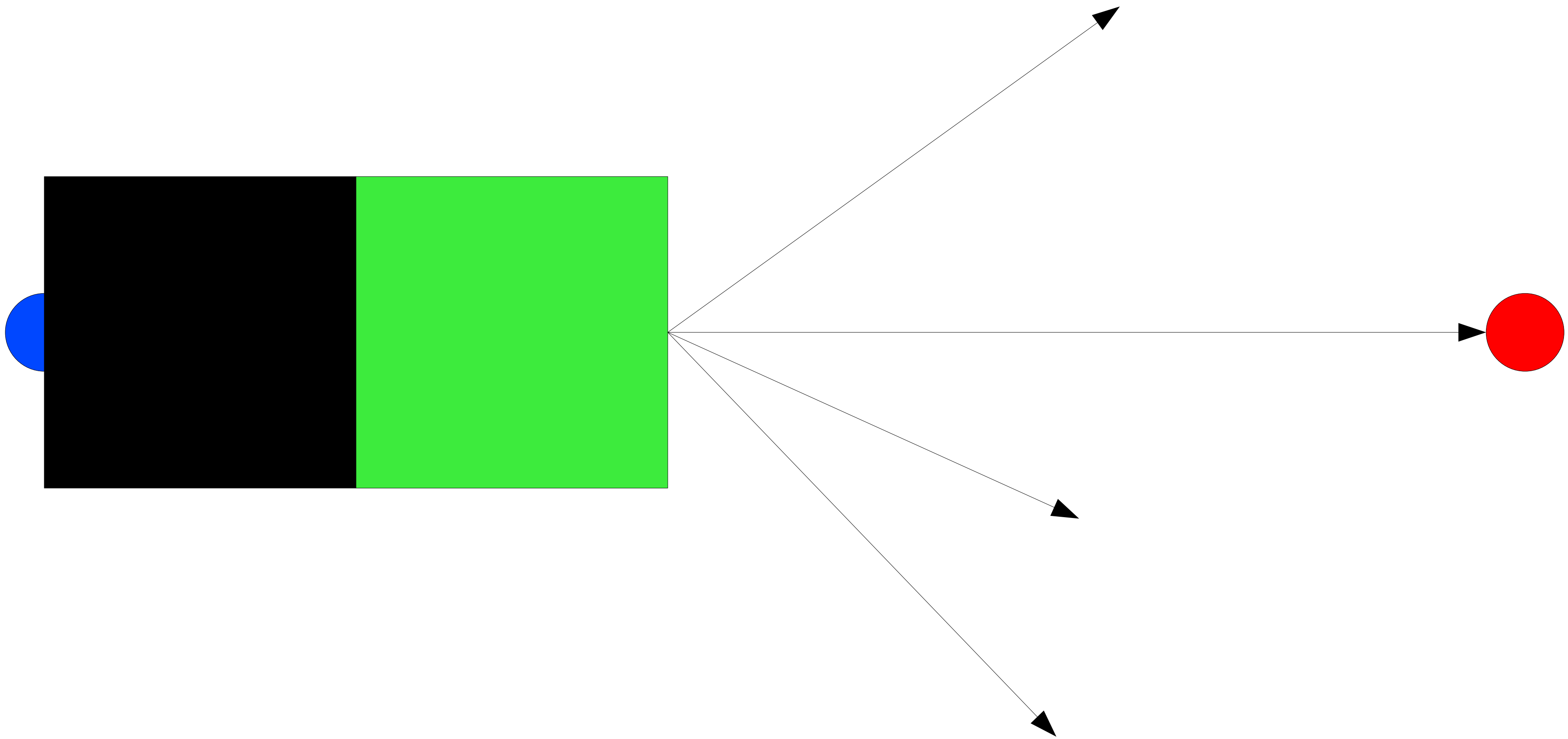
If live execution found, search deeper in candidate



Critical Transition Isolation

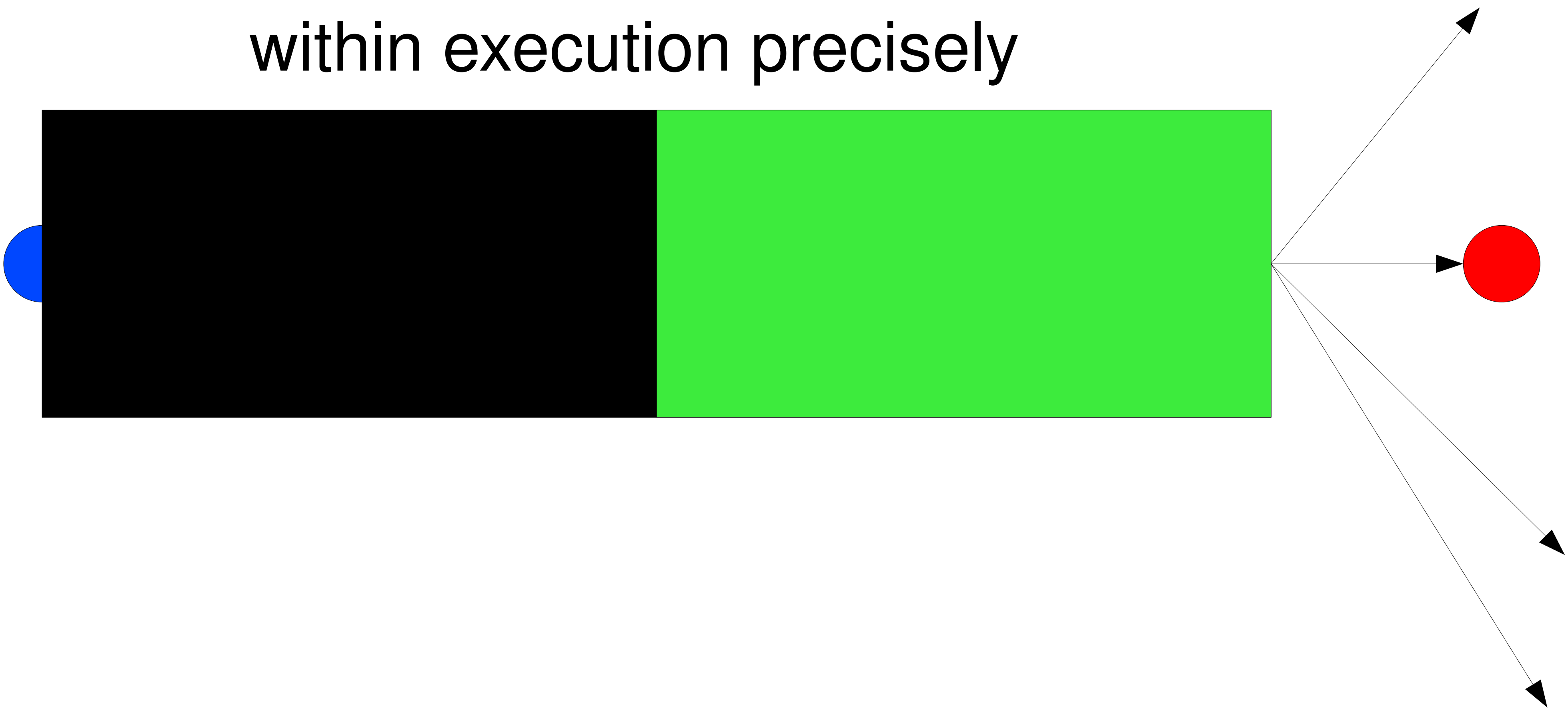


Critical Transition Isolation

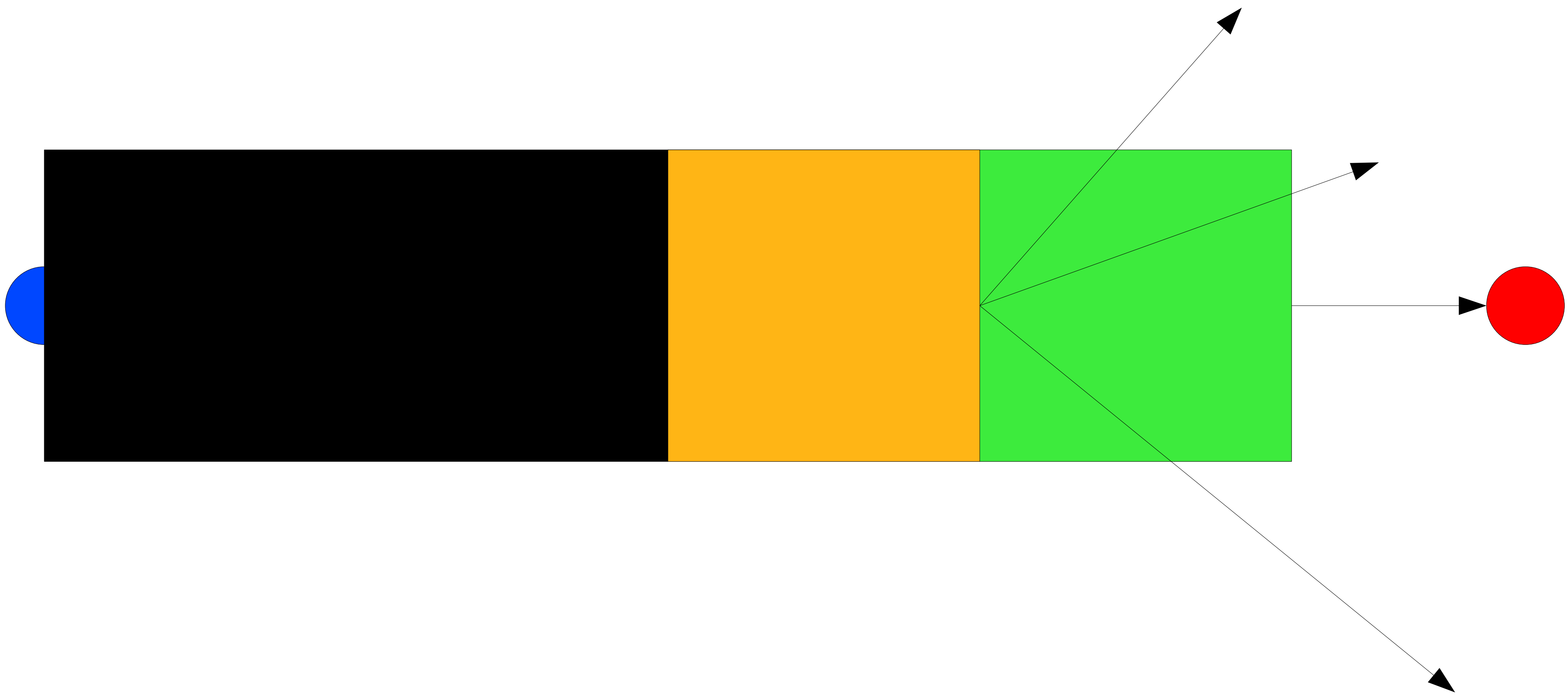


Critical Transition Isolation

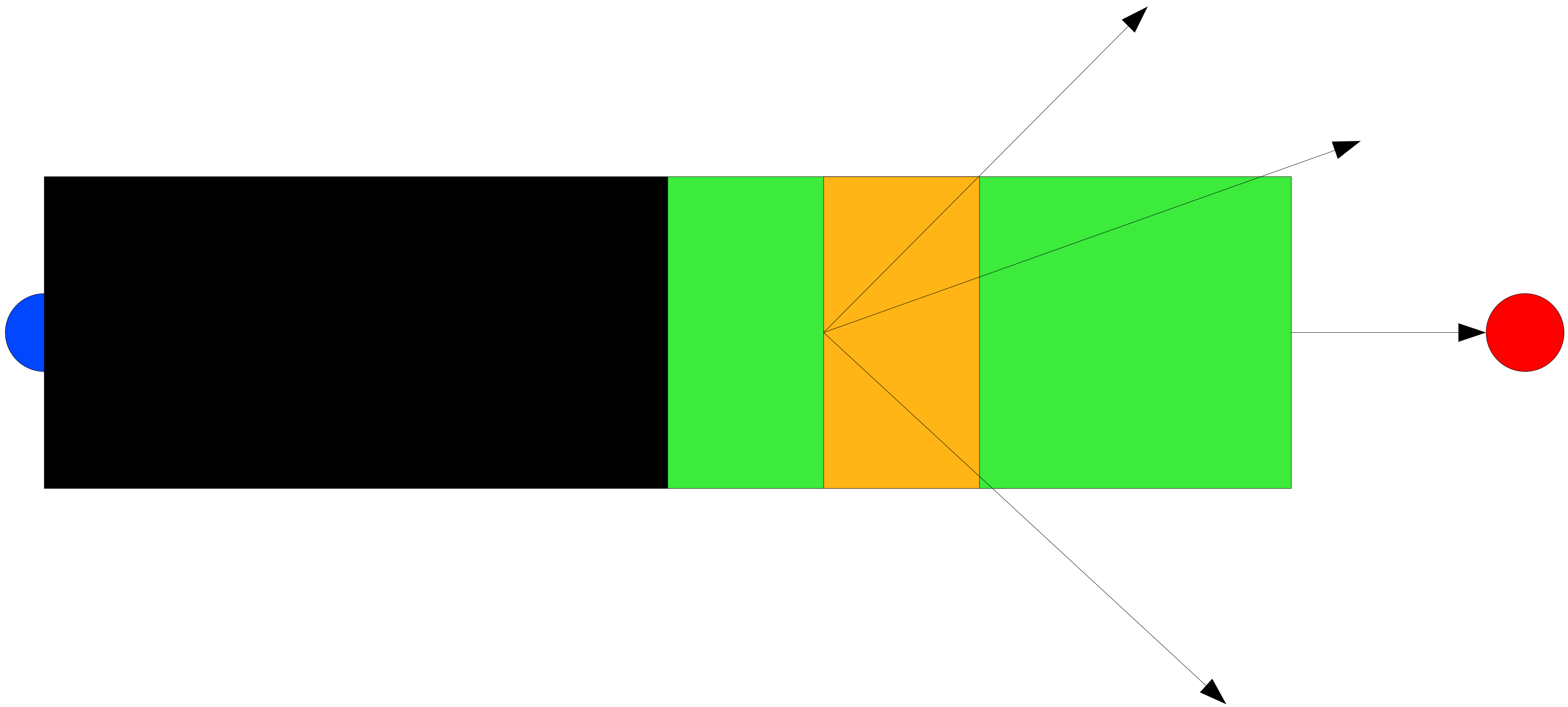
When dead state found, search back
within execution precisely



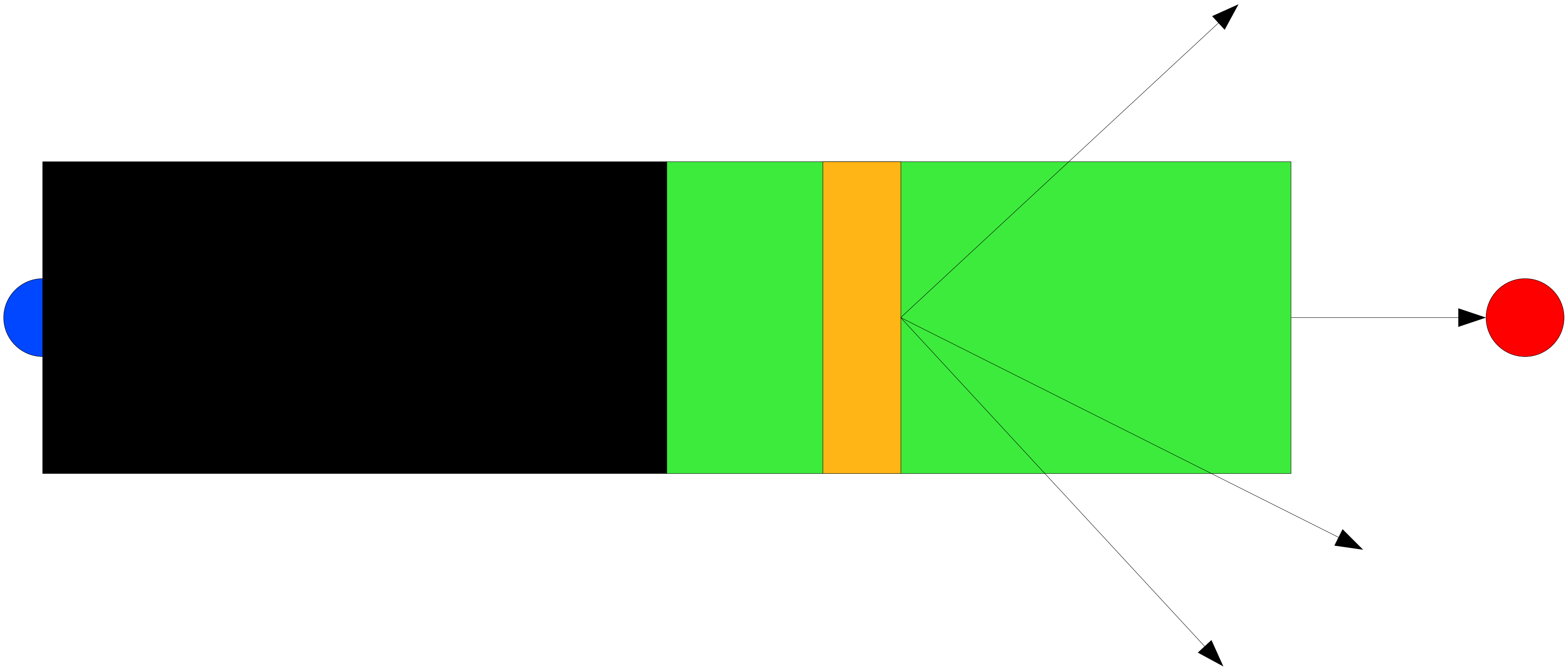
Critical Transition Isolation



Critical Transition Isolation

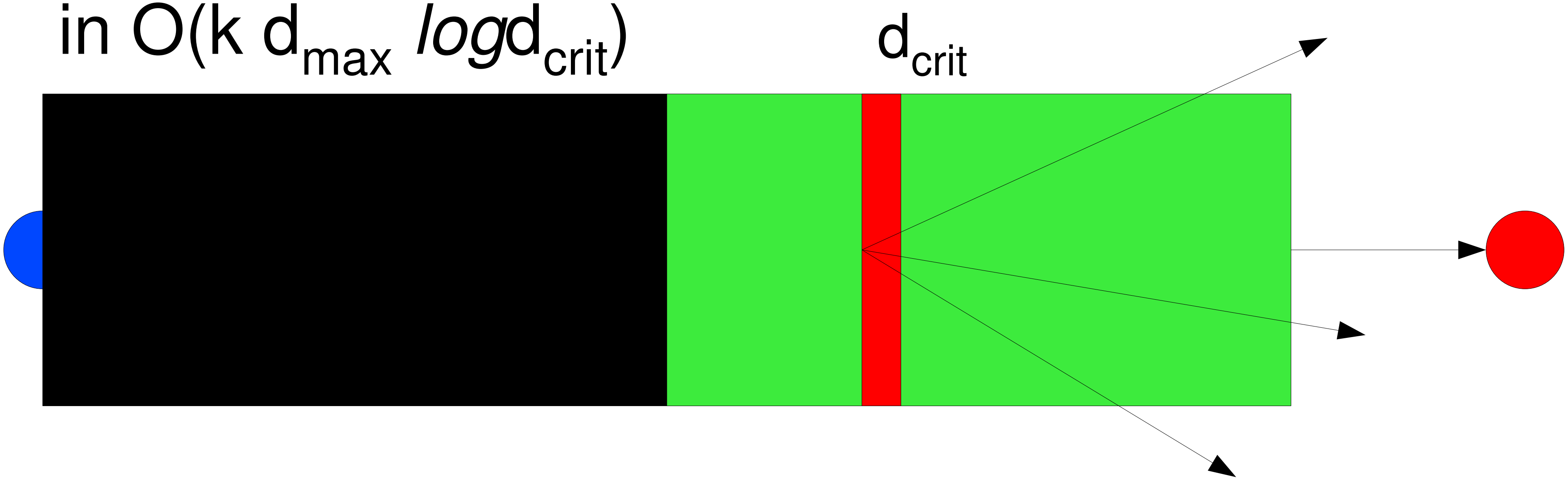


Critical Transition Isolation



Critical Transition Isolation

Eventually, critical transition is found in $O(k\ d_{max}\ \mathit{log}\, d_{crit})$



Observe, this also finds the longest common *live* prefix, which may help debug!

Process Errata

- Phase 1 of search may not find a dead state
  - The nature of *random* walks
  - May be transient violation

- Possible to find no initial live states; tune the parameters.

Implementation

MaceMC

- Replaces Mace C++ API for state machines w/ atomic handlers
- Requires mini driver creation for checking
- Assumes nondeterminism only through Mace API
- Timing model replaced via Mace API (logical or real)

State Explosion

Structured Transitions

- Mace is driven by atomic handlers
- Each handler is a coarse unit of simulation

State Explosion

- **State Hashing**- Hash state in order to recognize redundancies that needn't be explored
- **Stateless Search**- From initial state, reexecution is done by saving determinism decisions
- **Prefix-based Search**- To avoid initialization perturbations, wait until system reaches steady-state to search.

Biased Walks

- Reality does not provide a uniform distribution of (interesting) events.
- Randomly walk with bias towards realistic probabilities.
- Find live states sooner.
- Still reaches corner cases by exhaustive search.

Tuning

- **`k`** - # of random searched for liveness.
  - May be increased if false dead states found

- d_{max} – maximum random walk depth
  - May be tuned as with **`k`**.
  - Shows that exhaustive approaches are infeasible

MaceMC Debugger

- Critical Transition
- Reversible execution
- Exploring alternate paths
- Diff states
- Monitor events
- Message graph

Note: Logging space required in GBs

Testing

- Applied to 4 domains seen earlier
- Found same error/LOC as safety checkers
- Runtime: seconds to days

System	Bugs	Liveness	Safety	LOC
MaceTransport	11	5	6	585/3200
RandTree	17	12	5	309/2000
Pastry	5	5	0	621/3300
Chord	19	9	10	254/2200
Totals	52	31	21	

WiDS Checker: Combating Bugs in Distributed Systems

Xuezheng Liu, Wei Lin, Aimin Pan, Zheng Zhang

Goal

- For *reactive debugging* instead of model checking
- Execution is logged and replayed
- Predicate queries are applied over system execution.

A Common Problem

- “Evaluating the effectiveness of our tool is a challenge. The research community ... has not succeeded in producing a comprehensive set of benchmarks....”
- Applied to a handful of real bugs as in MaceMC.
- Identifies bugs at 'scale'

A Similar Playground

- Distribution API with runtime linkage for debugging and simulation.
- (Relatively) Atomic events form analysis units

But try to handle real world debugging issues

- (Modest) Scale
- Iterative debugging

Approach

- User queries are checked at event boundaries

  (timer, message, scheduler, synchronization) – via API

- Observed, logged events replayed in happens-before order on single system.
- Query scripts run over maintained state database
- Visualization and iterative replay/refinement

Replay

- Logging
  - All WiDS nondeterminism is logged
  - OS calls redirected and results captured to log
- Checkpointing
  - WiDS process context can be saved

Replay

- Start from beginning or checkpoint.
- Events replayed in serialized Lamport order
- Single process for simulation
  - Nodes are memory mapped files
  - Page table updates to support different processes
  - (Single node ~20 megs) ⇒ 40 nodes in 1 GB

Predicate Checking

- Values in database are refreshed after event
- Histories can be maintained
- Only modified predicates re-evaluated
- C++ types logged via compiler transform at allocation time.

Liveness

- Safety monitoring for liveness will cause false alarms
  - Additional derived variables are attached to predicates to allow filtering

```
declare_derived stabilized
begin_python
   retval = (Runtime.current_time - last_churn_time) / 10.0;
            if (retval < 1) : return retval;
   return 1;
end_python
                              # define predicates
predicate RingConsistency auxiliary stabilized{
   forall x in Node, exist y in Node,
      x.pred==y.id and y.succ == x.id
}
```

Testing

- Applied in 4 scenarios

Application	# of lines	# of bugs	Lines of script
Paxos	588	2	29
Lock server	2,439	2	33
BitVault	17,582	3	181
Macedon-chord	2,468	5	86