Shadow: Scalable Simulation for Systems Security Research

CrySP Speaker Series on Privacy

University of Waterloo

January 20th, 2016

Rob Jansen
U.S. Naval Research Laboratory
rob.g.jansen@nrl.navy.mil
@robgjansen

Talk Outline

- Shadow and how it works
- Tor research case study:
  Kernel-Informed Socket Transport
- Future directions

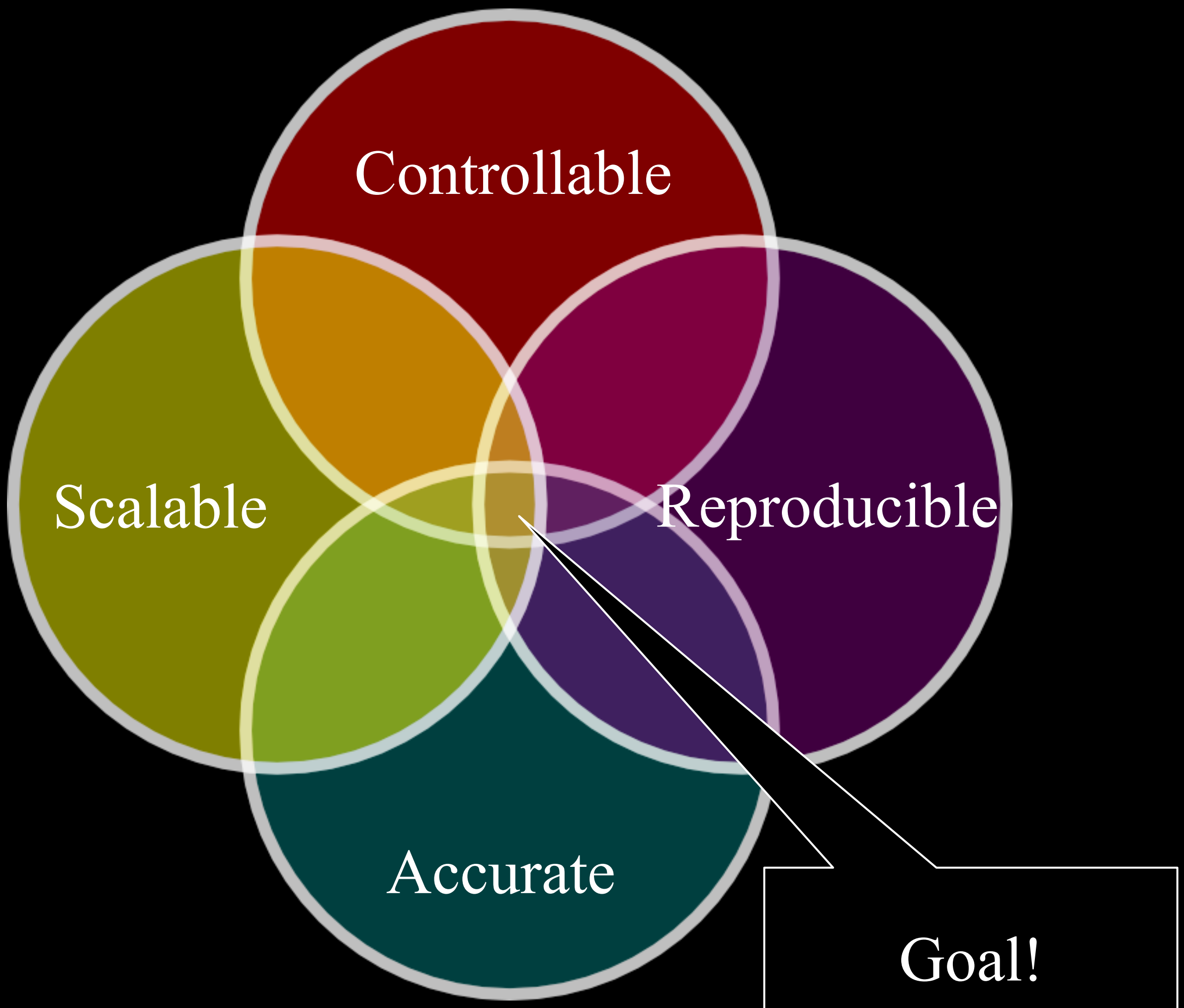
Why should you care?

- Expedite research and development
- Evaluate software mods or attacks without harming real users
- Understand holistic effects before deployment
- Shadow supports simulation for new applications

Thread 0

EXPERIMENTATION OPTIONS

Desirable Properties



Network Research Methods

Approaches	Problems
Live Network	Hard to manage, lengthy deployment, security risks
PlanetLab	Hard to manage, bad at modeling, not scalable
Simulation	Not generalizable, inaccurate
Emulation	Larger overhead, kernel complexities

Simulation vs Emulation

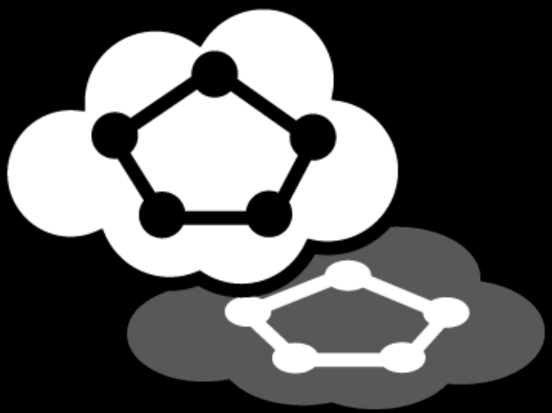
- Time (simulation wins)
  - Real time vs "as-fast-as-possible" execution
  - Emulation time must advance in synchrony with wall-clock time, or the virtual environment may become "sluggish" or unresponsive
  - Easier to slow down than to speed up execution!
- Realism (emulation wins)
  - Uses host OS kernel, protocols, applications
  - Can run anything that runs on OS

Thread 1

SHADOW

What is Shadow?

- Parallel discrete-event network simulator
- Models routing, latency, bandwidth
- Simulates time, CPU, OS
  - TCP/UDP, sockets, queuing, threading
- Emulates POSIX C API on Linux
- Directly executes apps as plug-ins

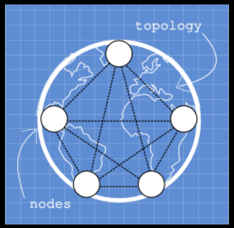


Simulation Environment

Hosts

Logical processing units with independent state

Simulation Environment



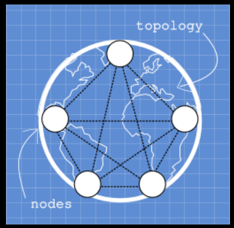
Hosts

Network

Routing elements (nodes, links) and attributes (bandwidth, latency, packet loss)

Simulation Environment

Hosts



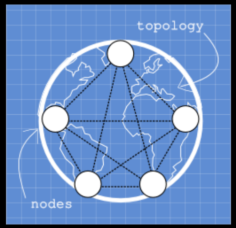
Network

Global Clock

Holds current virtual time (distinct from physical time)

Simulation Environment

Hosts



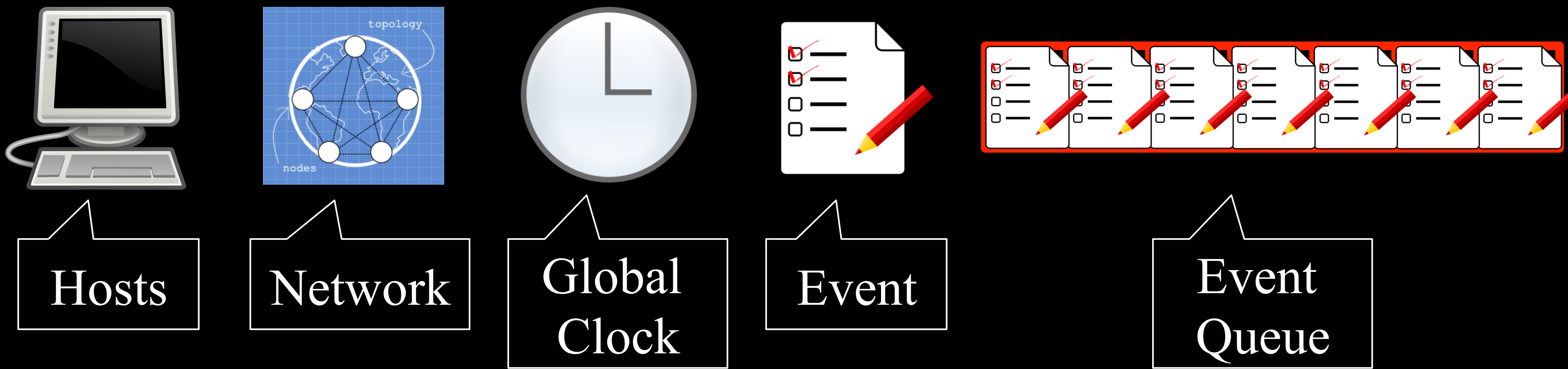
Network

Global Clock

Event

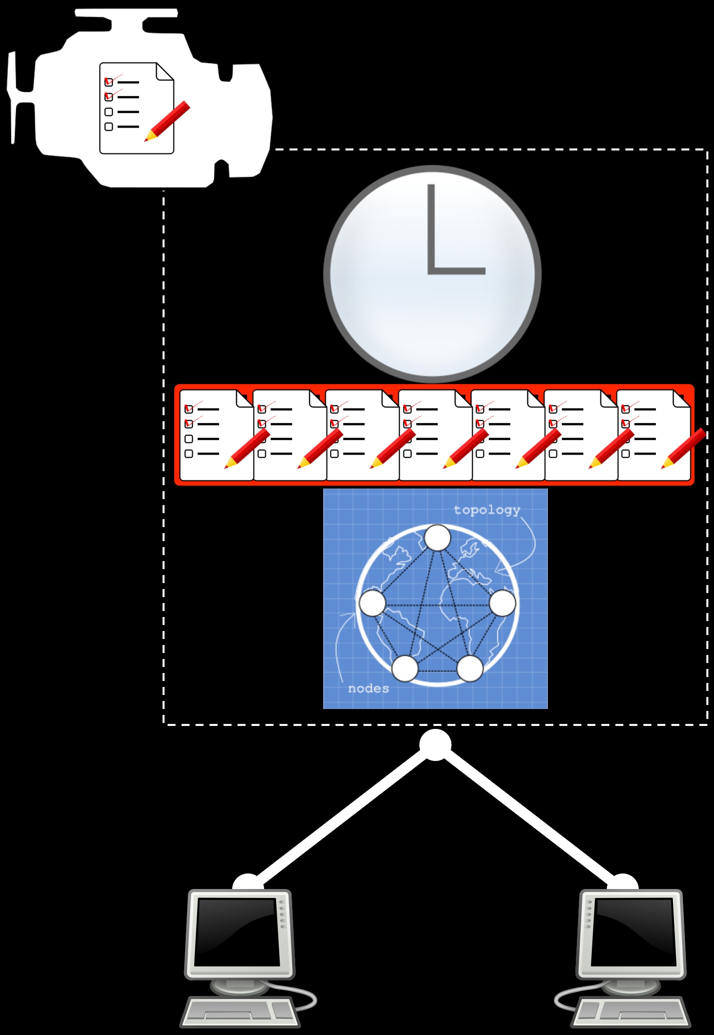
Processing task for a host at a specific time

Simulation Environment



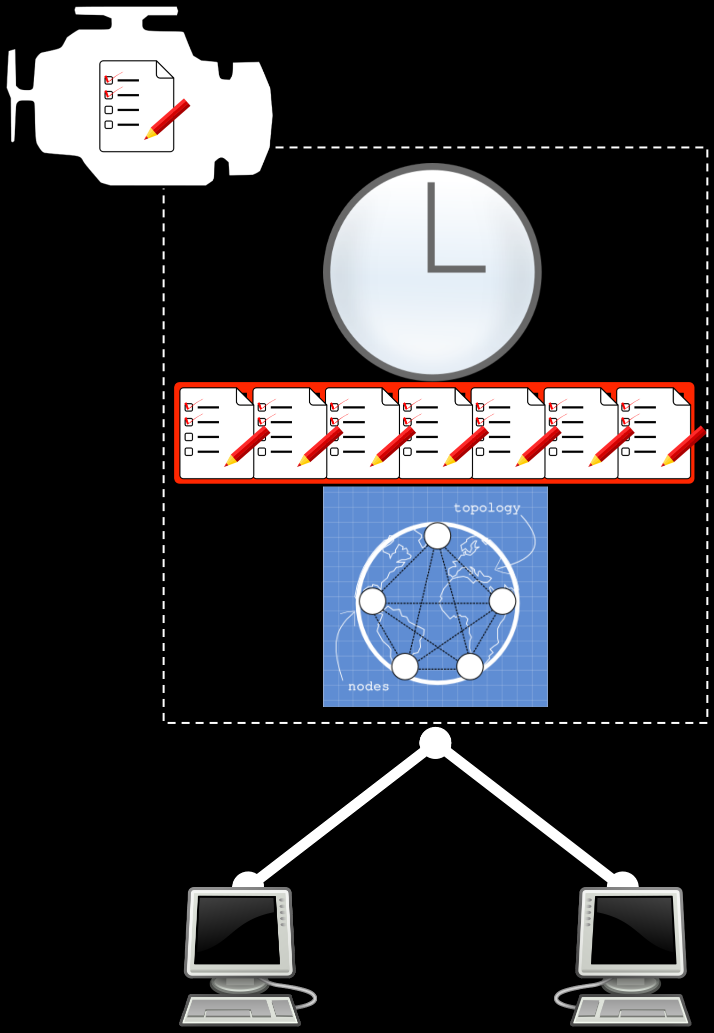
Holds events sorted by time (min heap)

Discrete Event Engine



- Facilitate communication: exchange events between hosts through the network
- "as-fast-as-possible" execution

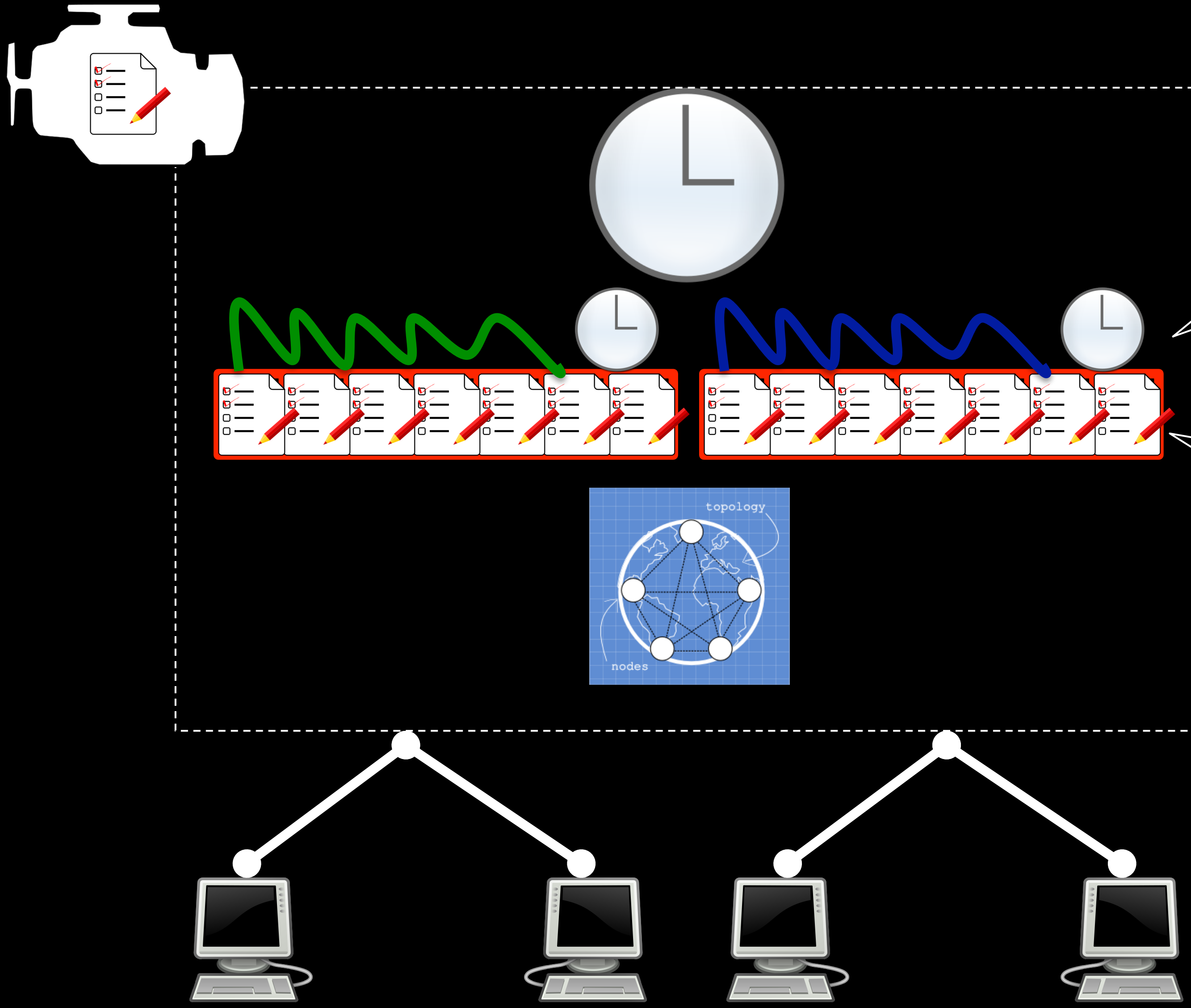
Discrete Event Engine



- Facilitate communication: exchange events between hosts through the network
- "as-fast-as-possible" execution

- While !end
  - Get next event
  - Update clock
  - Process event
    - Enqueue events

Parallel Discrete Event Engine



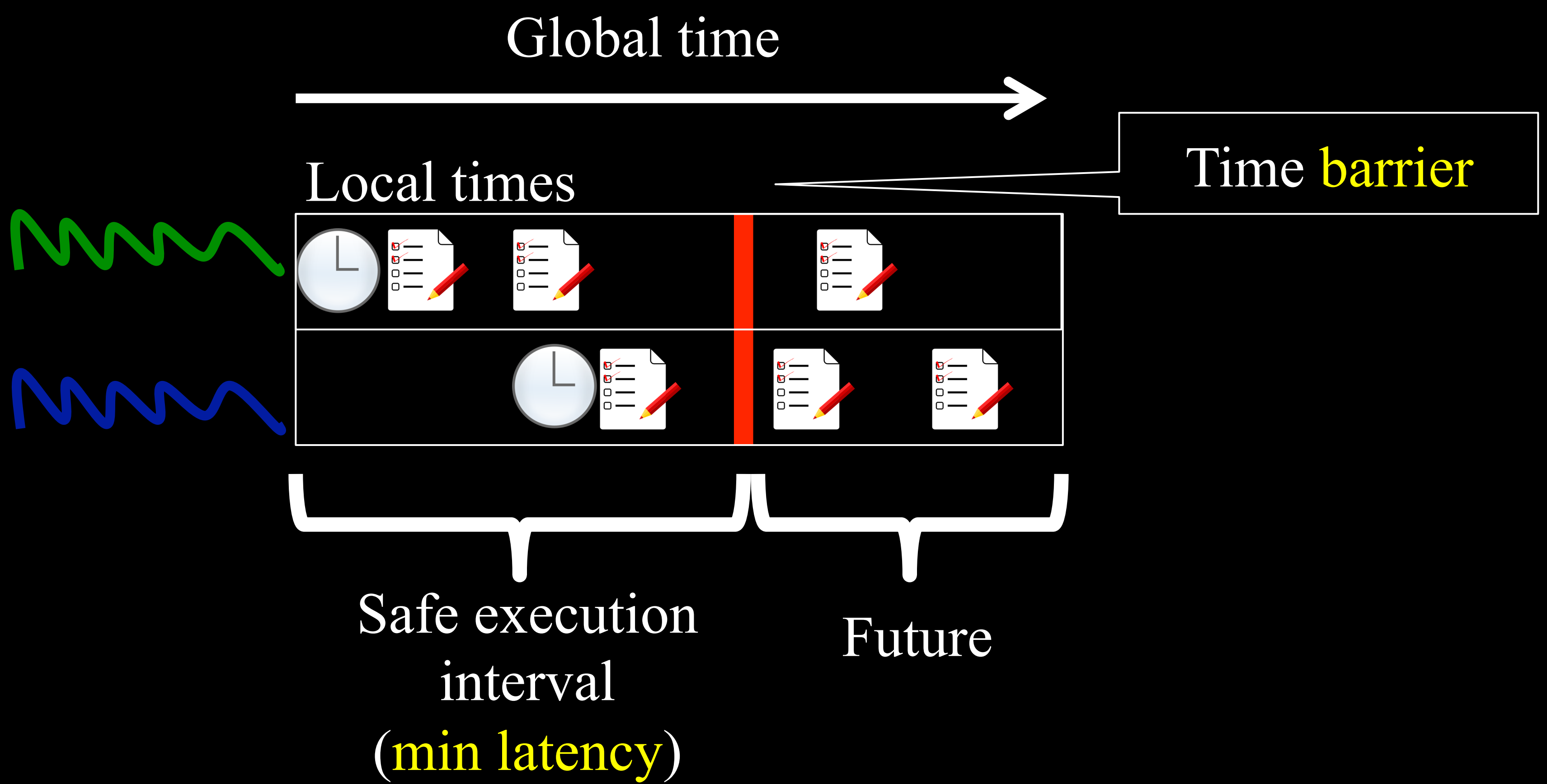
Host workloads split among physical threads

Each physical thread has event queue

Synchronization problem!

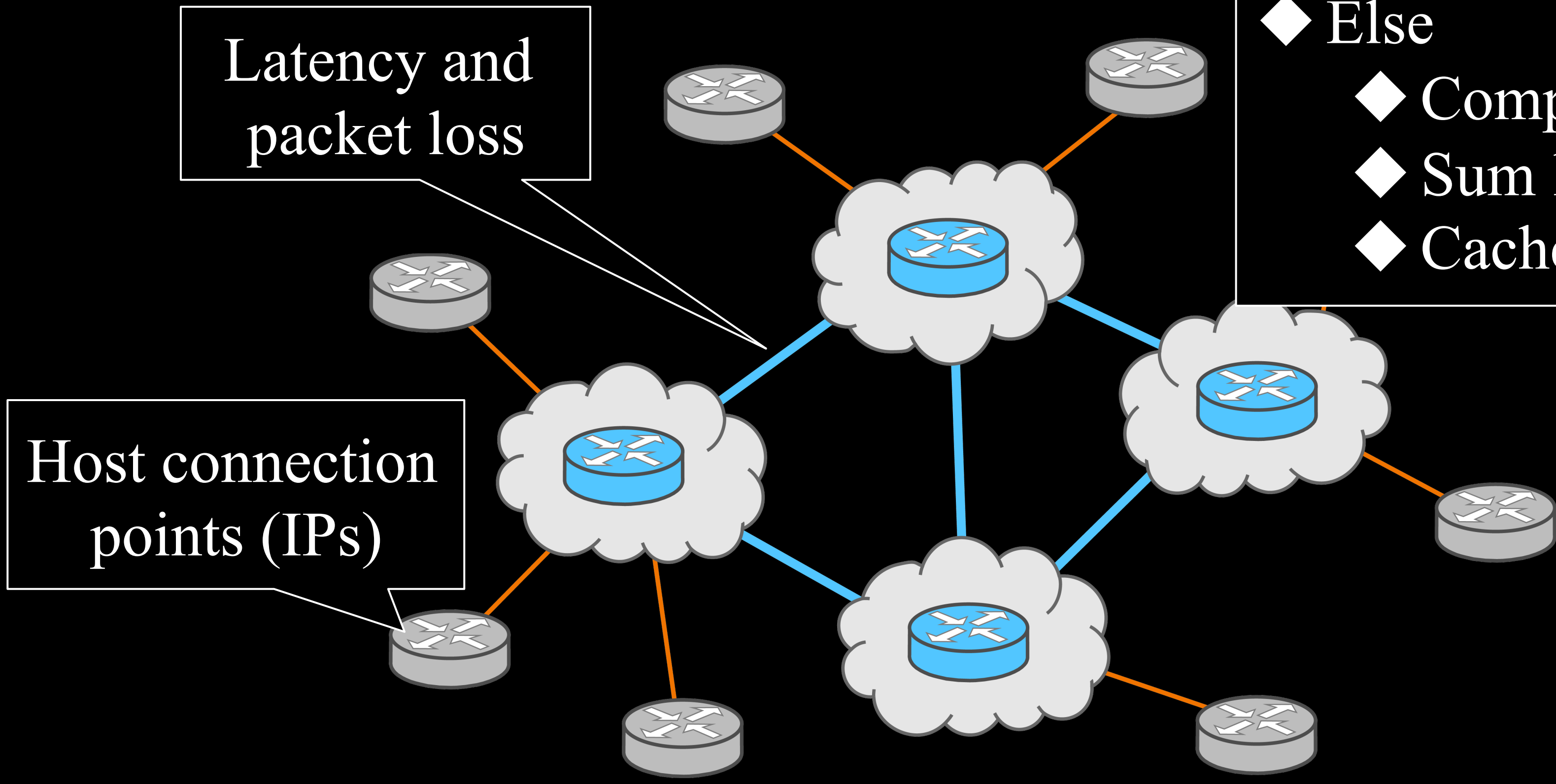
Conservative Synchronization

- Ensure causality
  - events must occur in correct order (not in the past)



Virtual Network Routing

- Network graph model



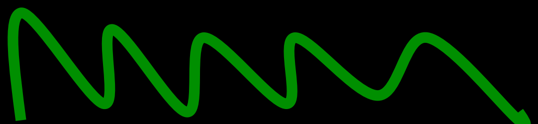
- If complete:
  - Lookup link
  - Get latency
- Else
  - Compute shortest path
  - Sum link latencies
  - Cache result

Executing Applications on Hosts

- Load programs as dynamic shared object libraries

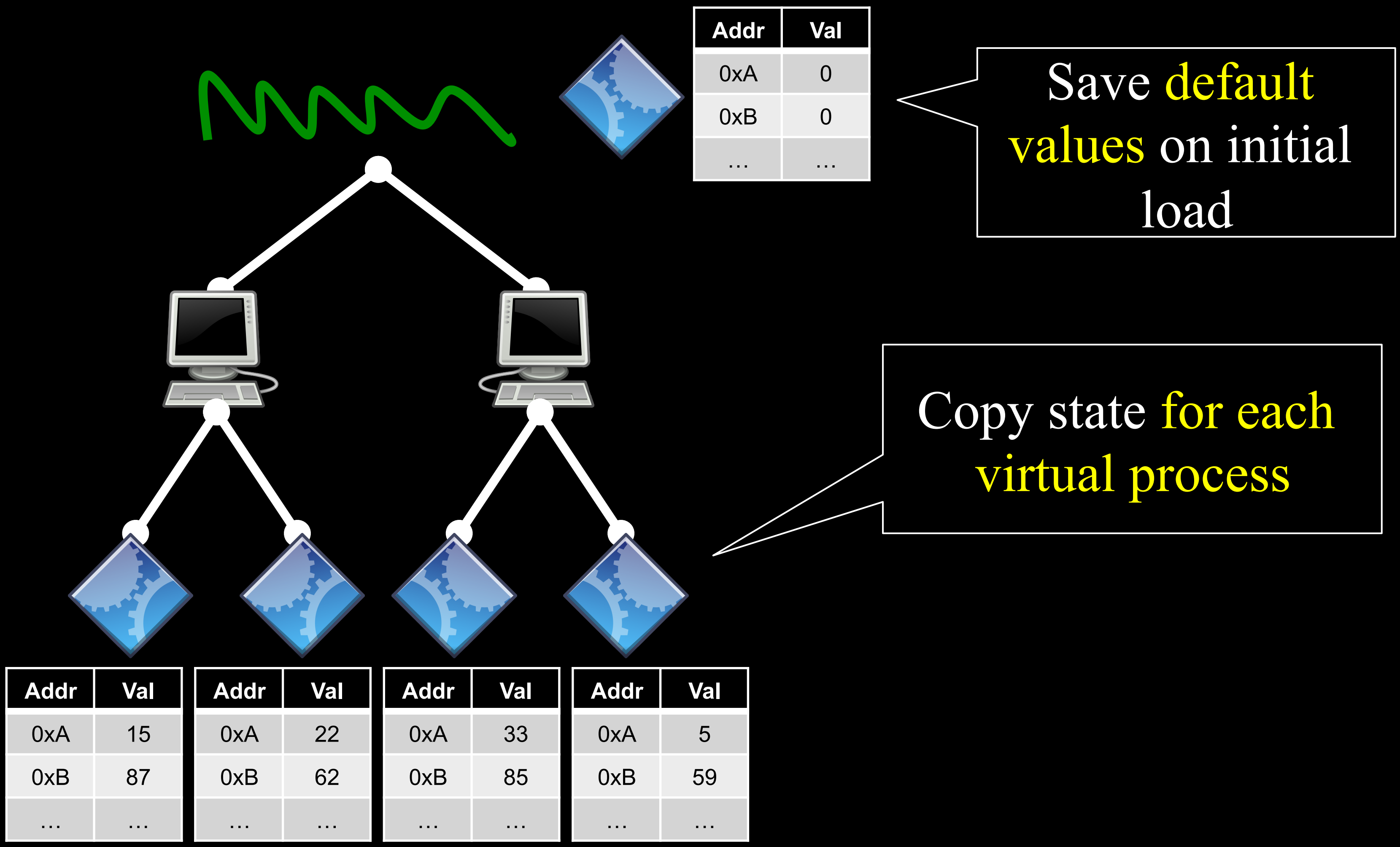
Addr	Val
0xA	0
0xB	0
…	…

Compile with Clang, extract state addresses with LLVM pass



Each program loaded only once per thread

Virtual Process Management



Addr	Val
0xA	0
0xB	0
…	…

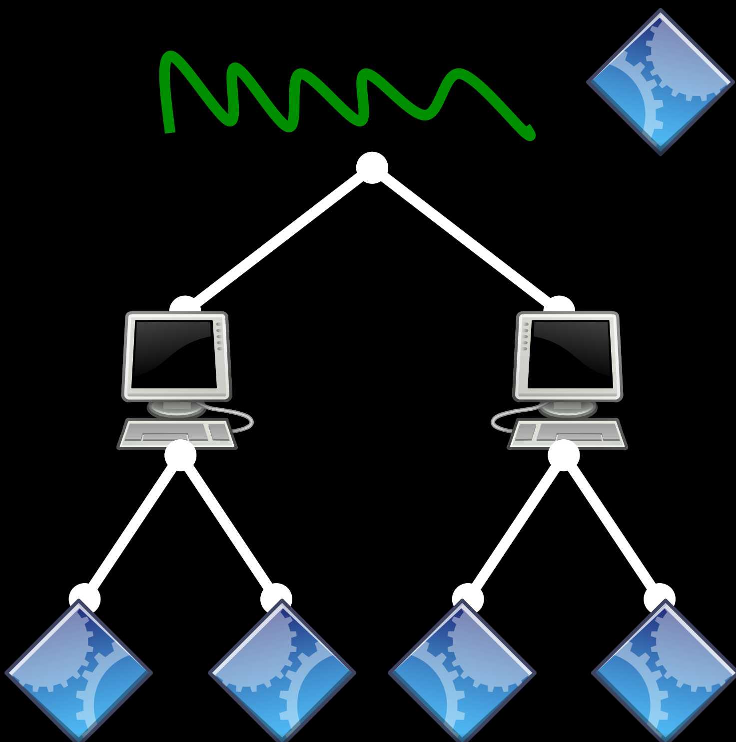
Addr	Val
0xA	15
0xB	87
…	…

Addr	Val
0xA	22
0xB	62
…	…

Addr	Val
0xA	33
0xB	85
…	…

Addr	Val
0xA	5
0xB	59
…	…

Virtual Process Management



Addr	Val
0xA	0
0xB	0
…	…

Addr	Val
0xA	15
0xB	87
…	…

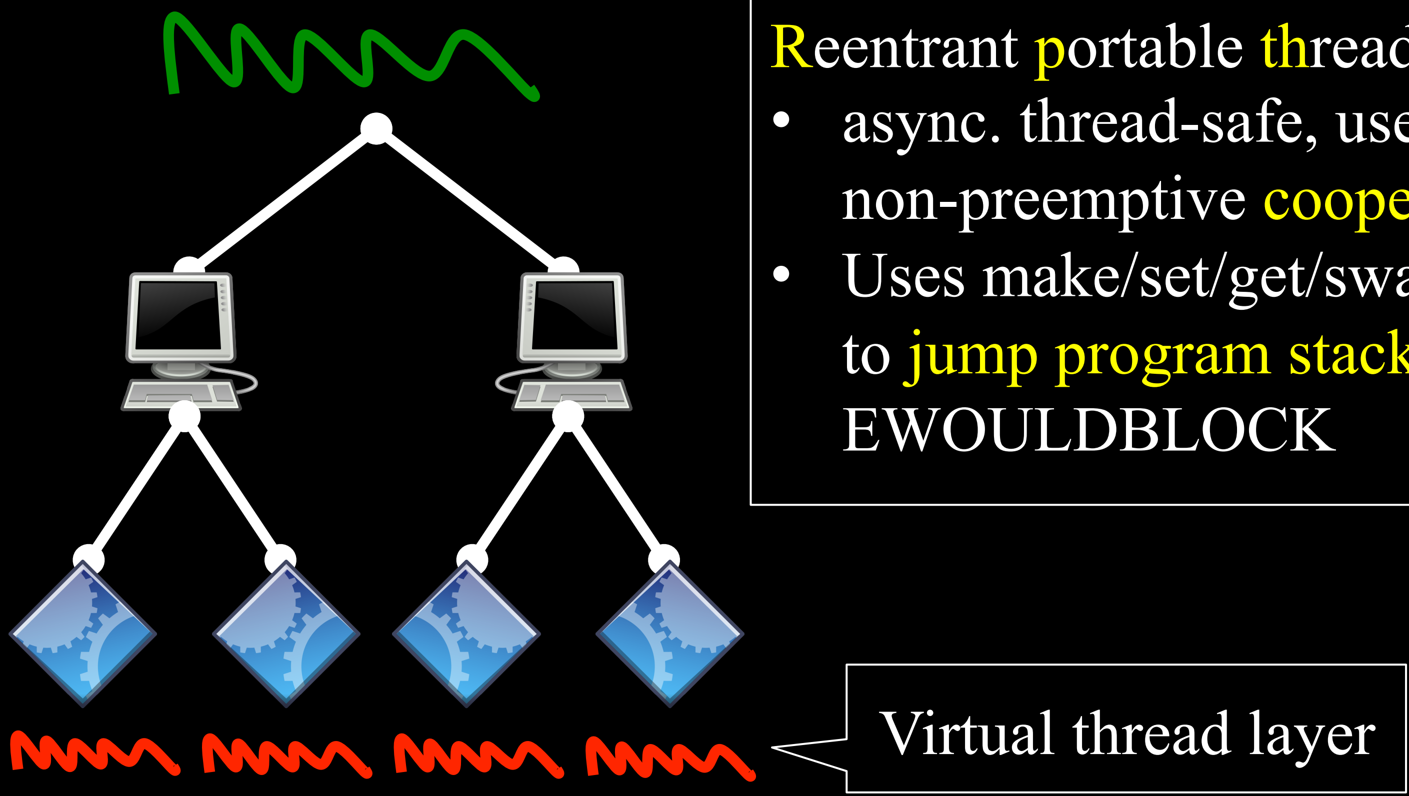
Addr	Val
0xA	22
0xB	62
…	…

Addr	Val
0xA	33
0xB	85
…	…

Addr	Val
0xA	5
0xB	59
…	…

Swap state into/out of memory as virtual processes are switched

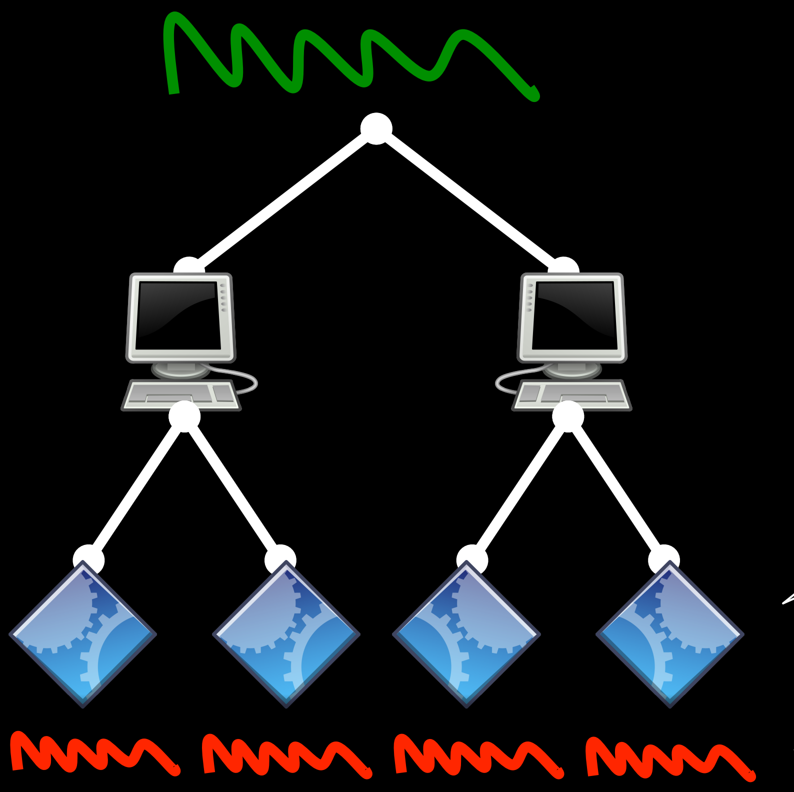
Virtual Thread Management



Reentrant portable threads (rpth)

- async. thread-safe, user-land non-preemptive cooperative threading
- Uses make/set/get/swapcontext() magic to jump program stacks when EWOULDBLOCK

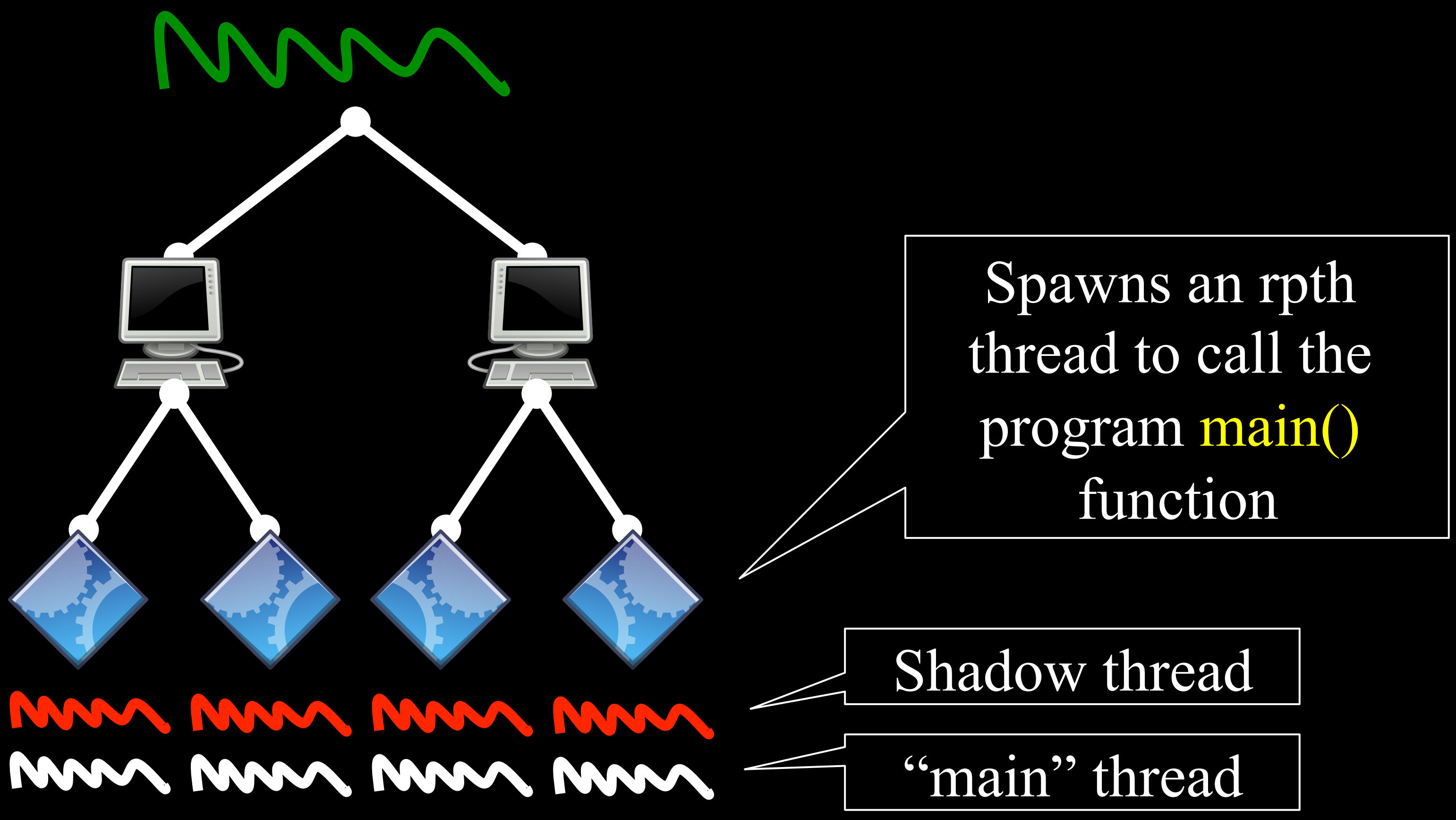
Virtual Thread Management



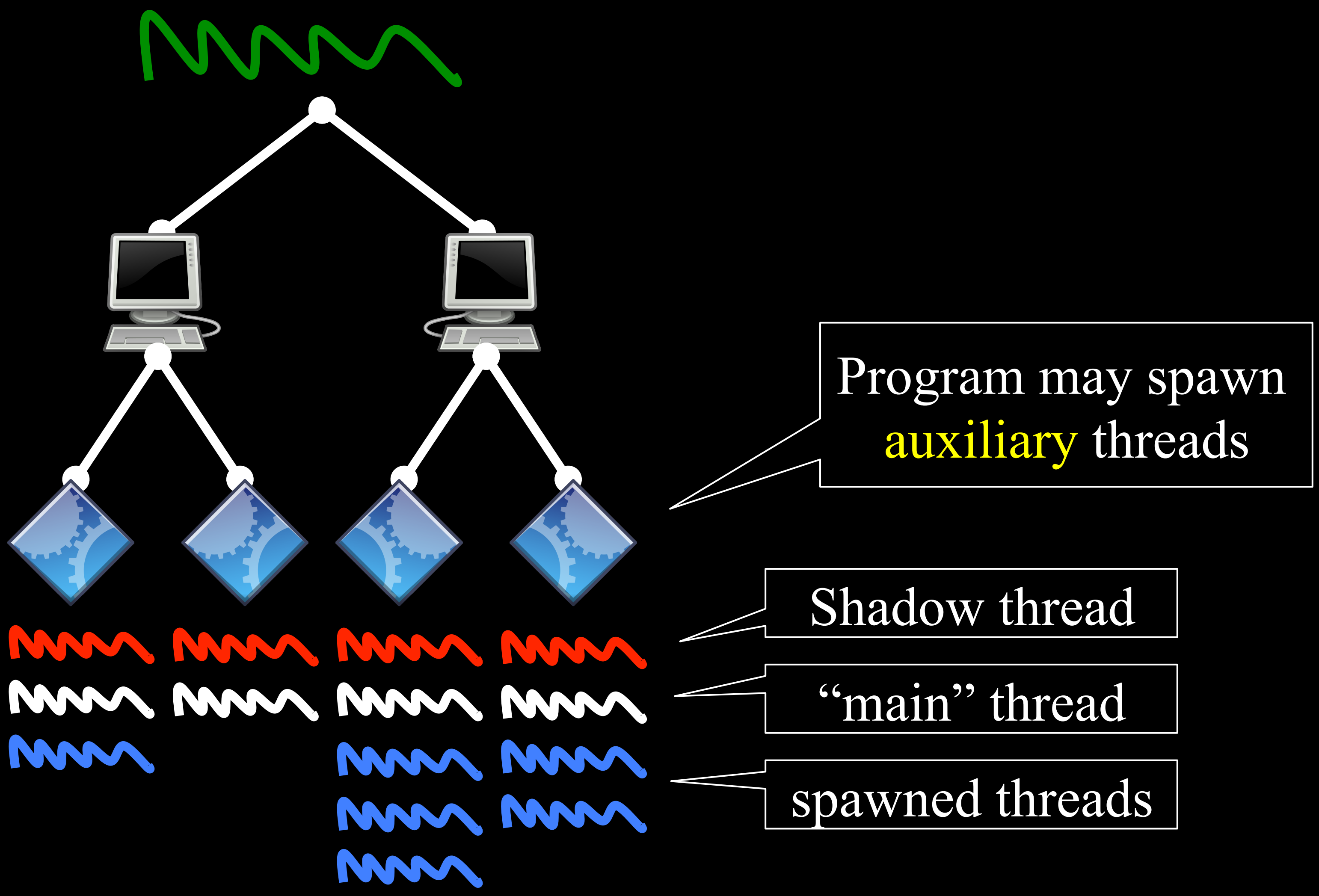
Each virtual process has a private rpth instance

Shadow thread

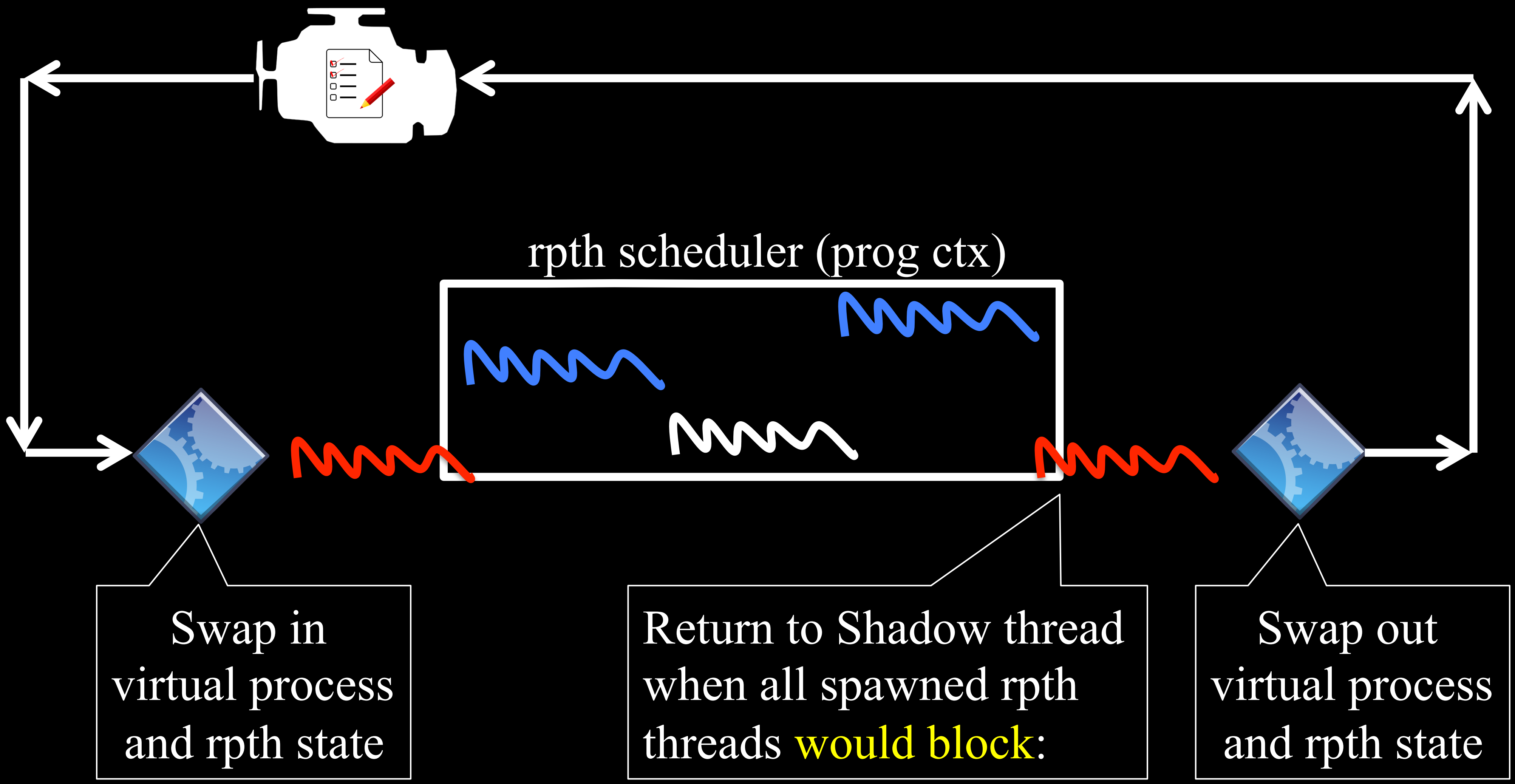
Virtual Thread Management



Virtual Thread Management



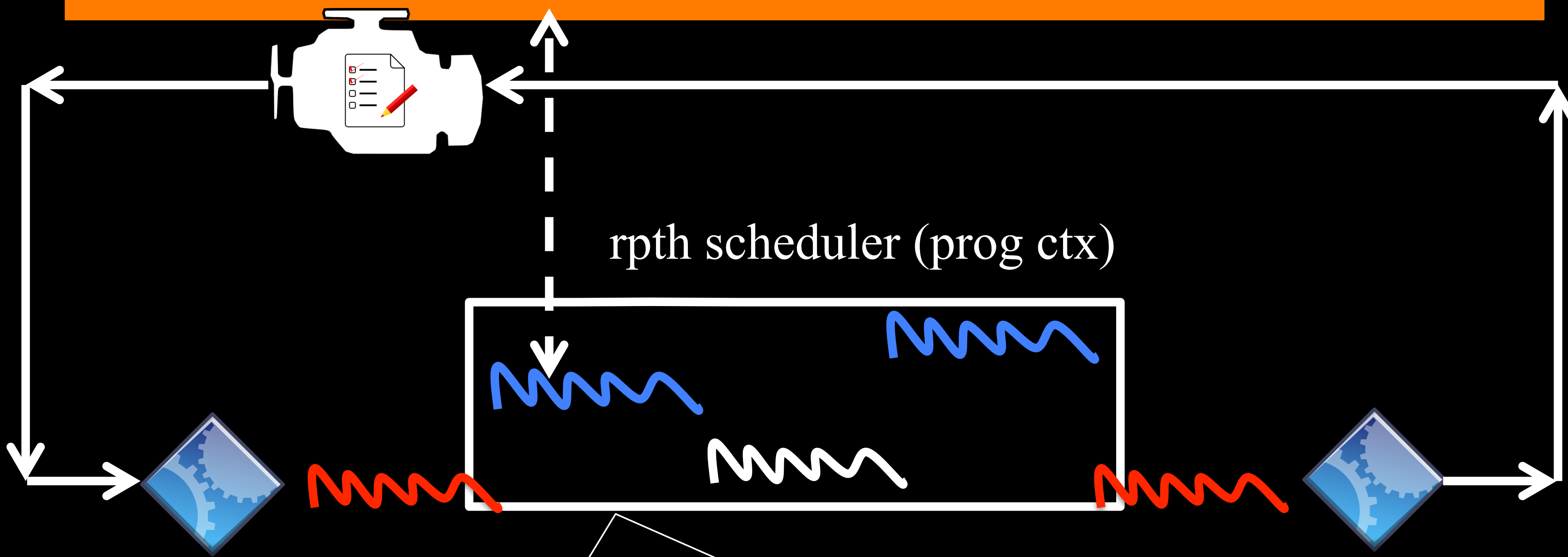
Execution Flow with rpth



Function Interposition

App
Libraries
(libc, …)

LD_PRELOAD=libpreload.so (*socket, write, pthread_create, …*)



Function calls are redirected
to simulated counterpart

Simulating a Kernel

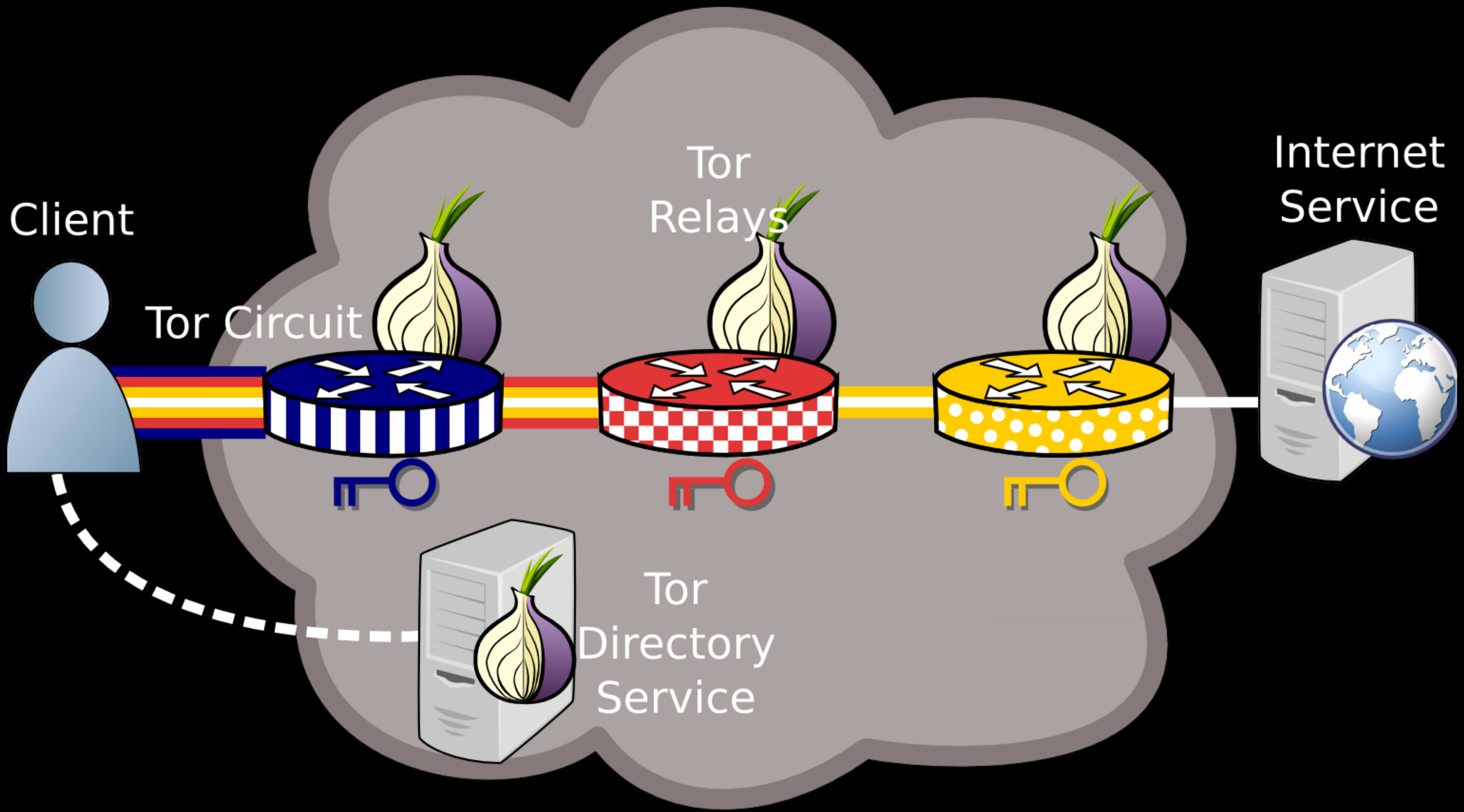
- Sockets and queuing
- Network protocols – TCP, UDP
- Threading (pthread)
- Randomization (maintain determinism)
- CPU usage

Thread 2

KERNEL INFORMED SOCKET TRANSPORT

With John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson

Anonymous Communication: Tor



This Talk

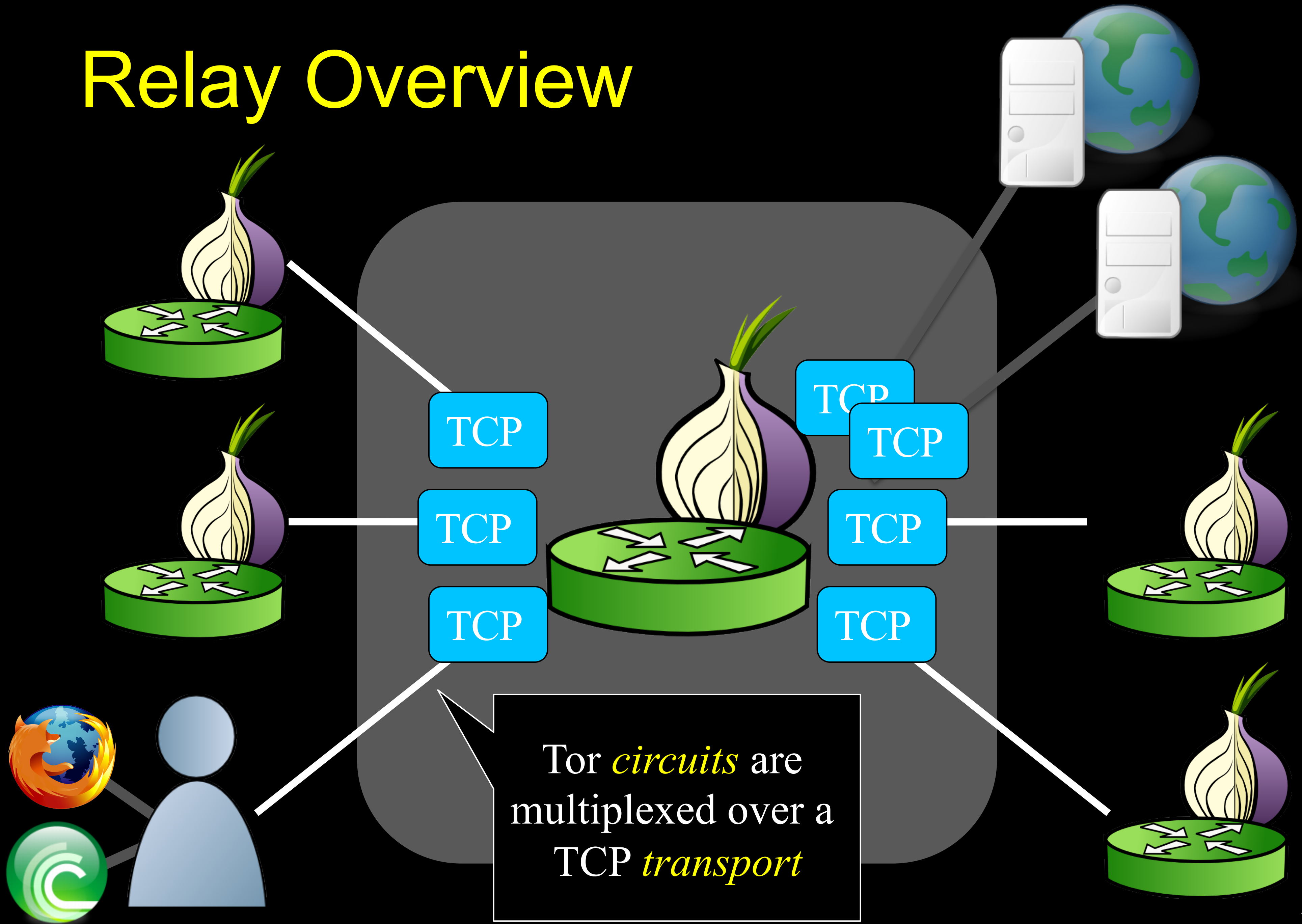
- Where is Tor slow?
  - Measure public Tor and private Shadow-Tor networks
  - Identify circuit scheduling and socket flushing problems
- Design KIST: Kernel-Informed Socket Transport
  - Use TCP snd_cwnd to limit socket writes
- Evaluate KIST Performance and Security
  - Reduces kernel and end-to-end circuit congestion
  - Throughput attacks unaffected, speeds up latency attacks

Outline

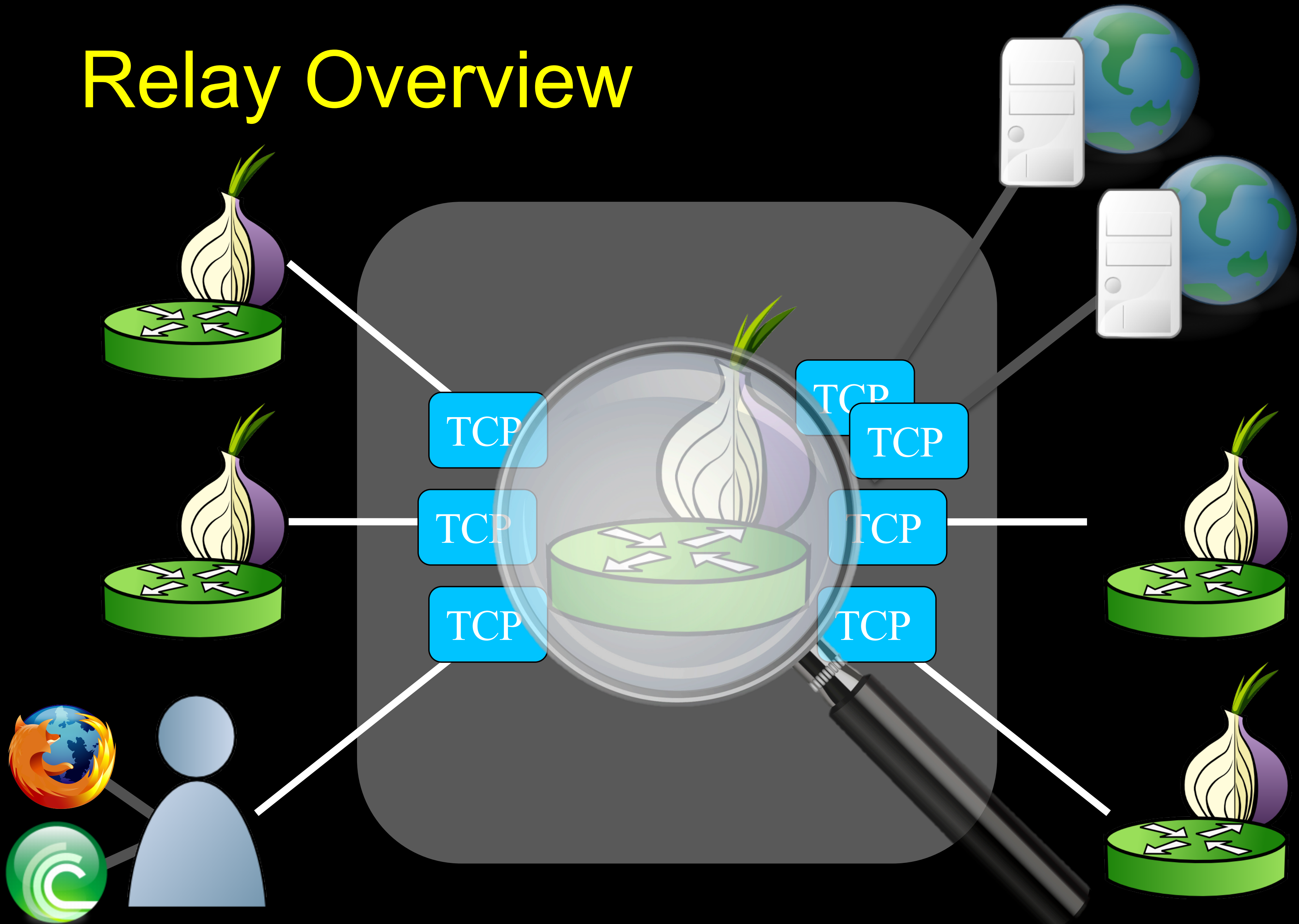
- Background
- Instrument Tor, measure congestion
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security

Relay Overview

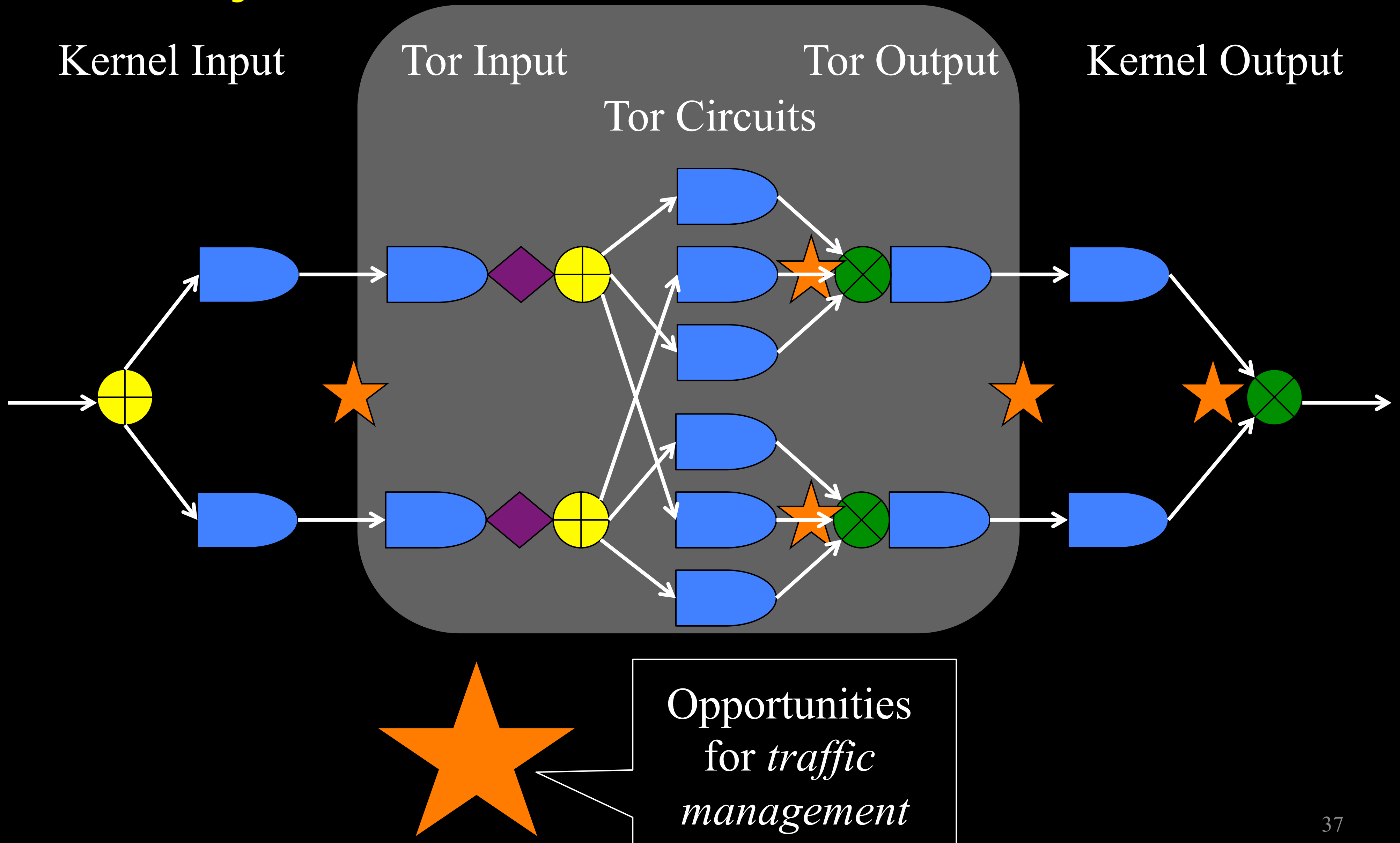
Relay Overview



Relay Overview



Relay Internals

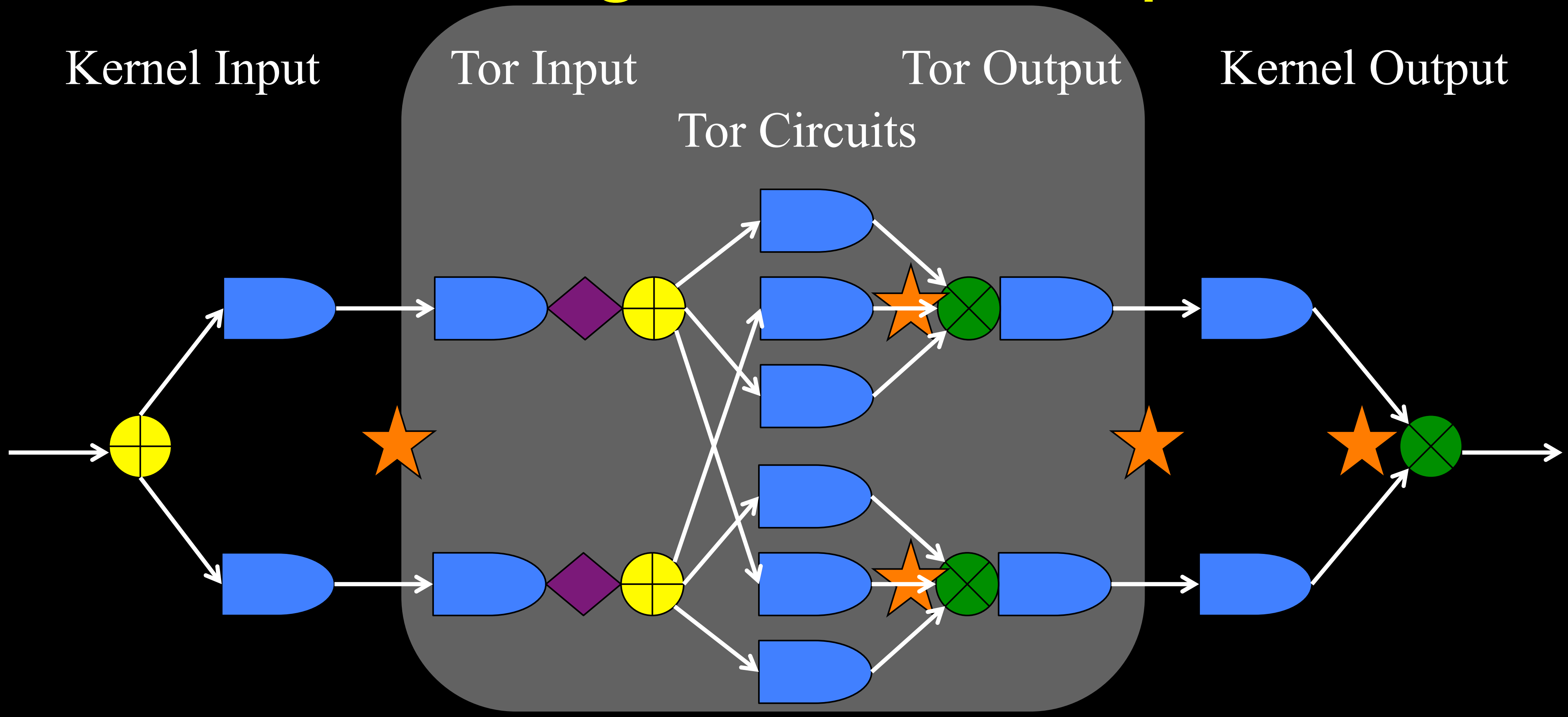


Opportunities for *traffic management*

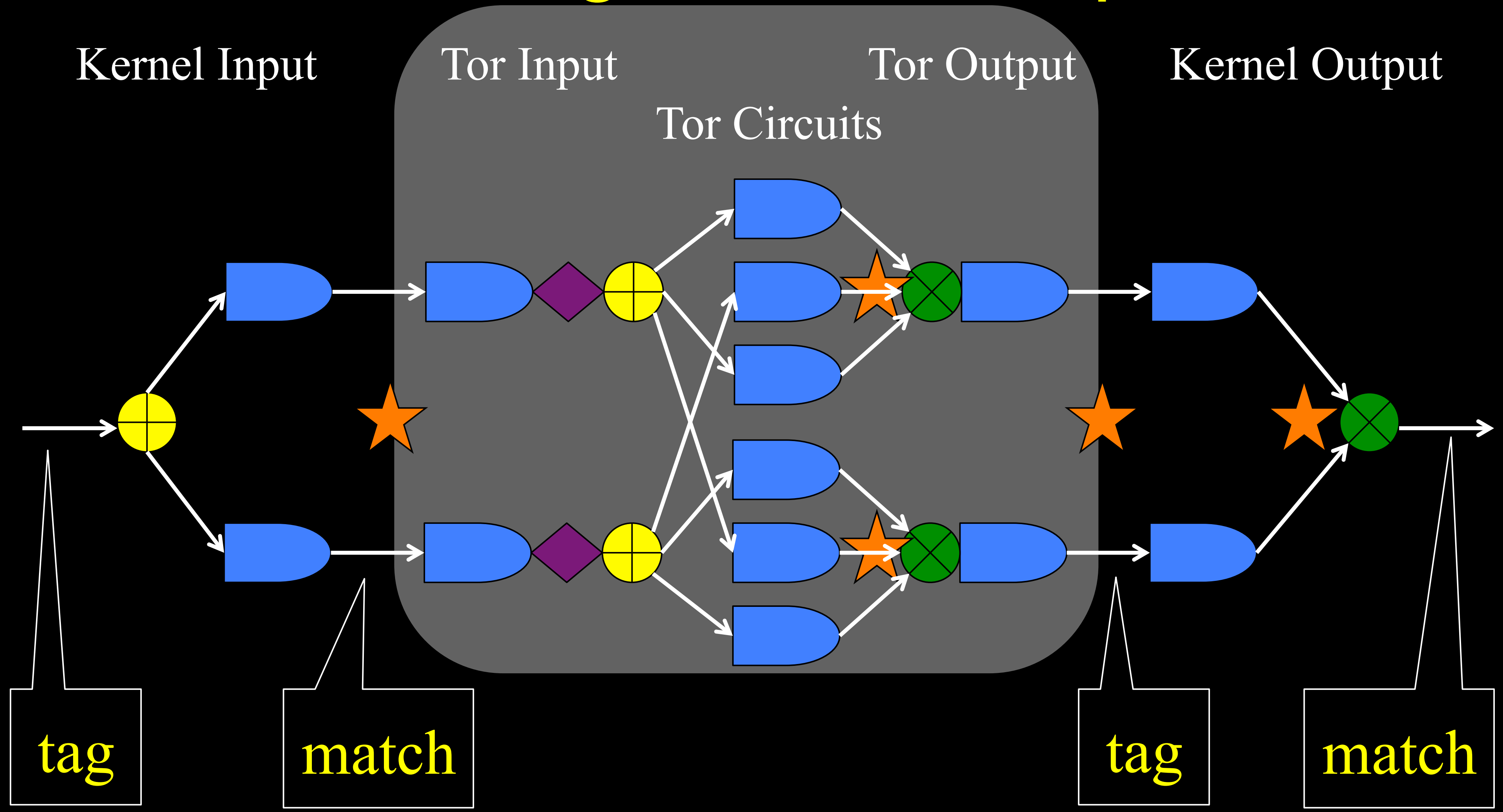
Outline

- ~~Background~~
- Instrument Tor, measure congestion
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security

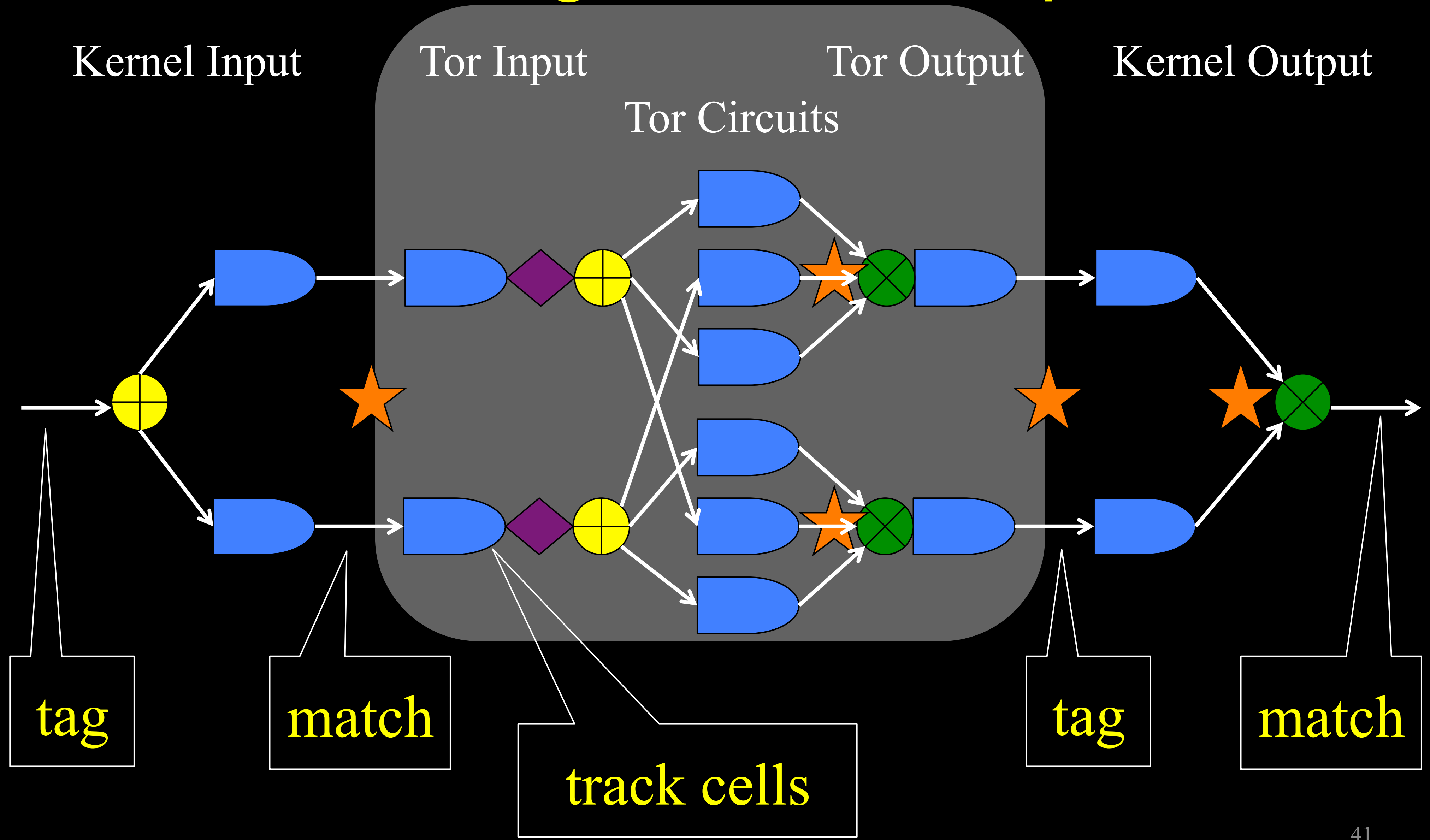
Live Tor Congestion - libkqtime



Live Tor Congestion - libkqtime



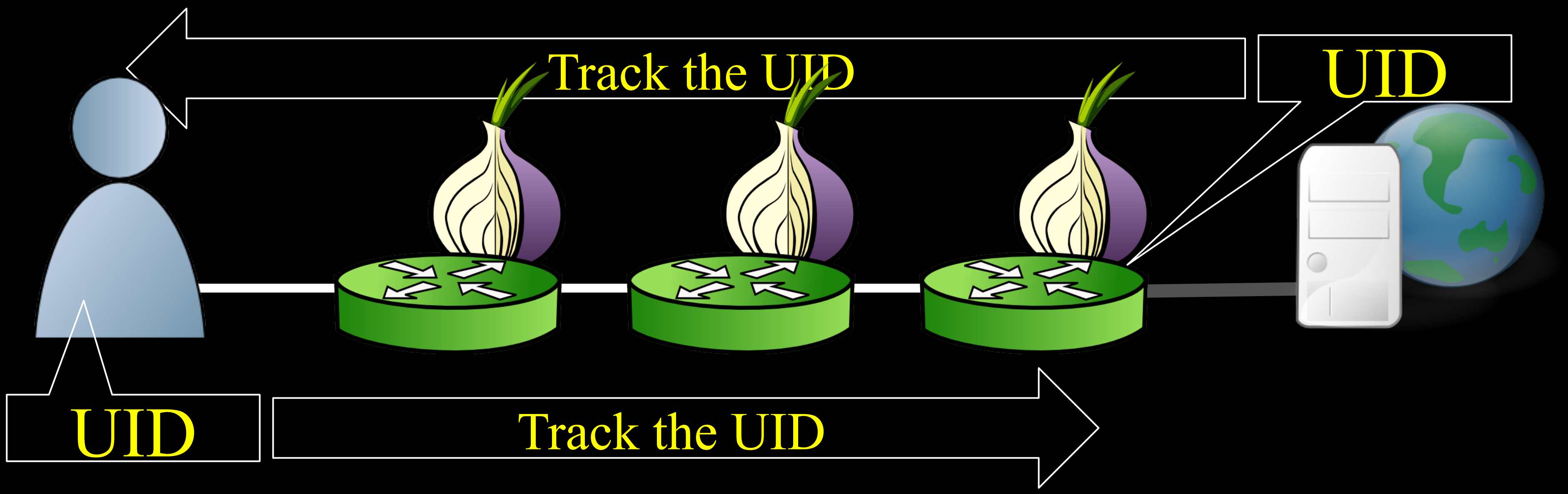
Live Tor Congestion - libkqtime



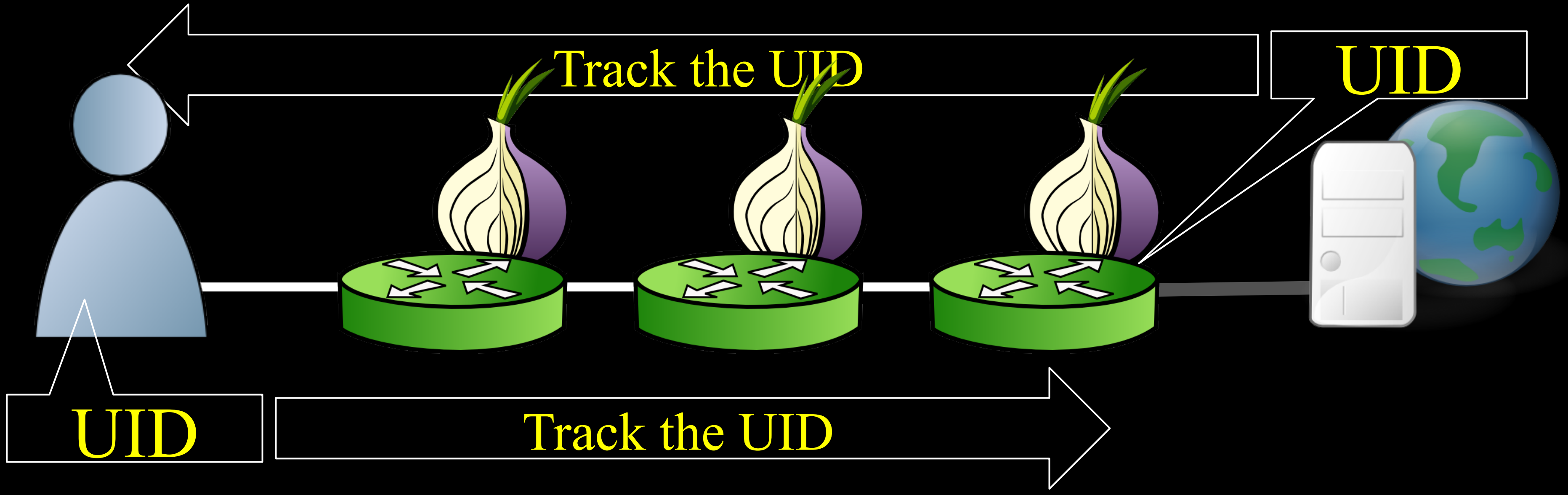
Shadow Network Simulation

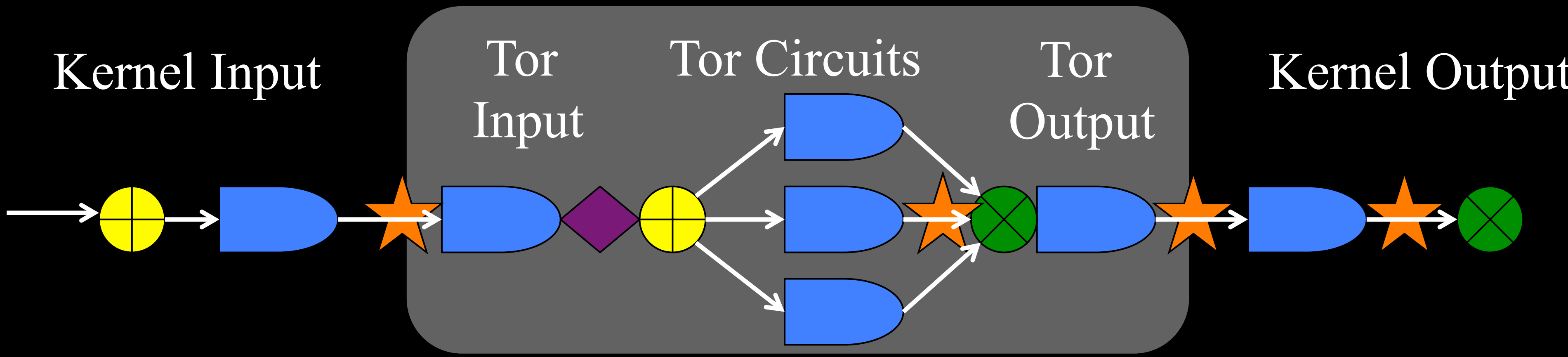
- Enhanced Shadow with several missing TCP algorithms
  - CUBIC congestion control
  - Retransmission timers
  - Selective acknowledgements (SACK)
  - Forward acknowledgements (FACK)
  - Fast retransmit/recovery
- Designed largest known private Tor network
  - 3600 relays and 12000 simultaneously active clients
  - Internet topology graph: ~700k nodes and 1.3m links

Shadow-Tor Congestion – UIDs

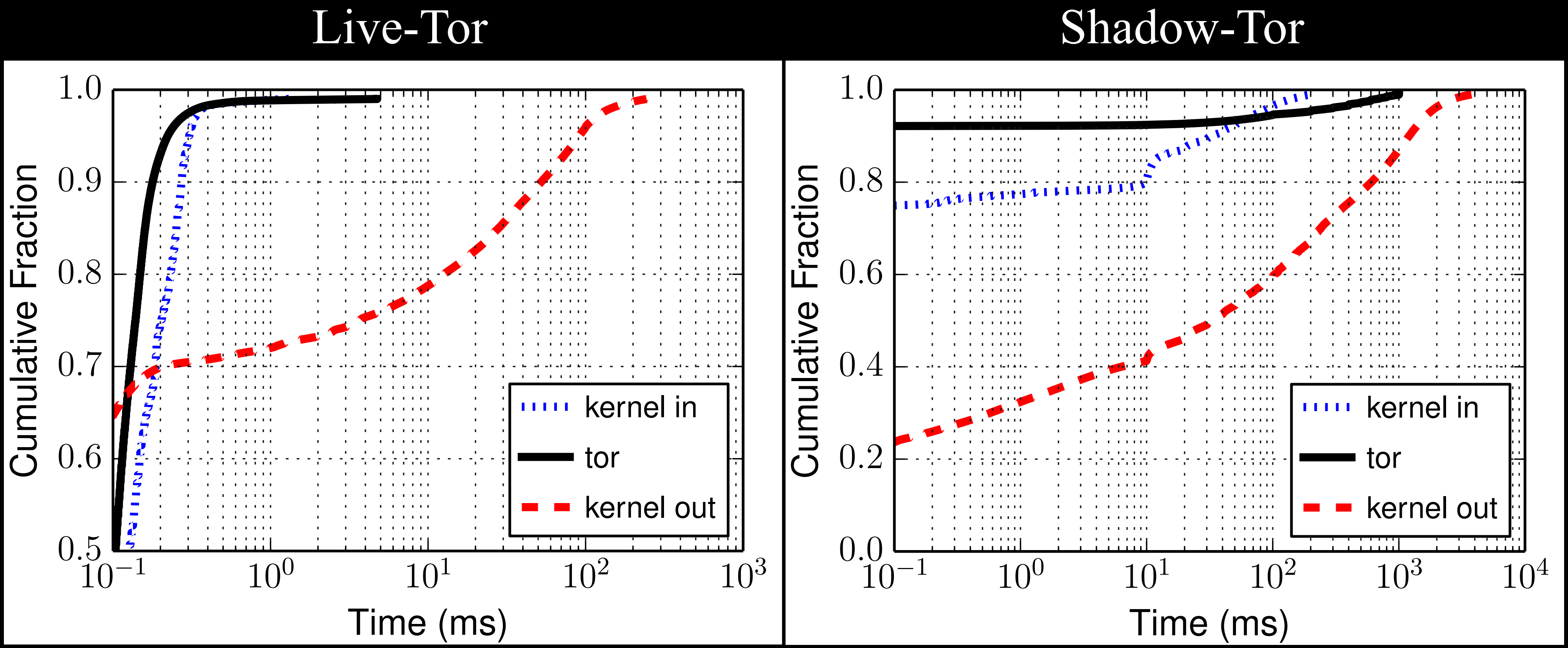


Shadow-Tor Congestion – UIDs





Tor and Shadow-Tor Congestion

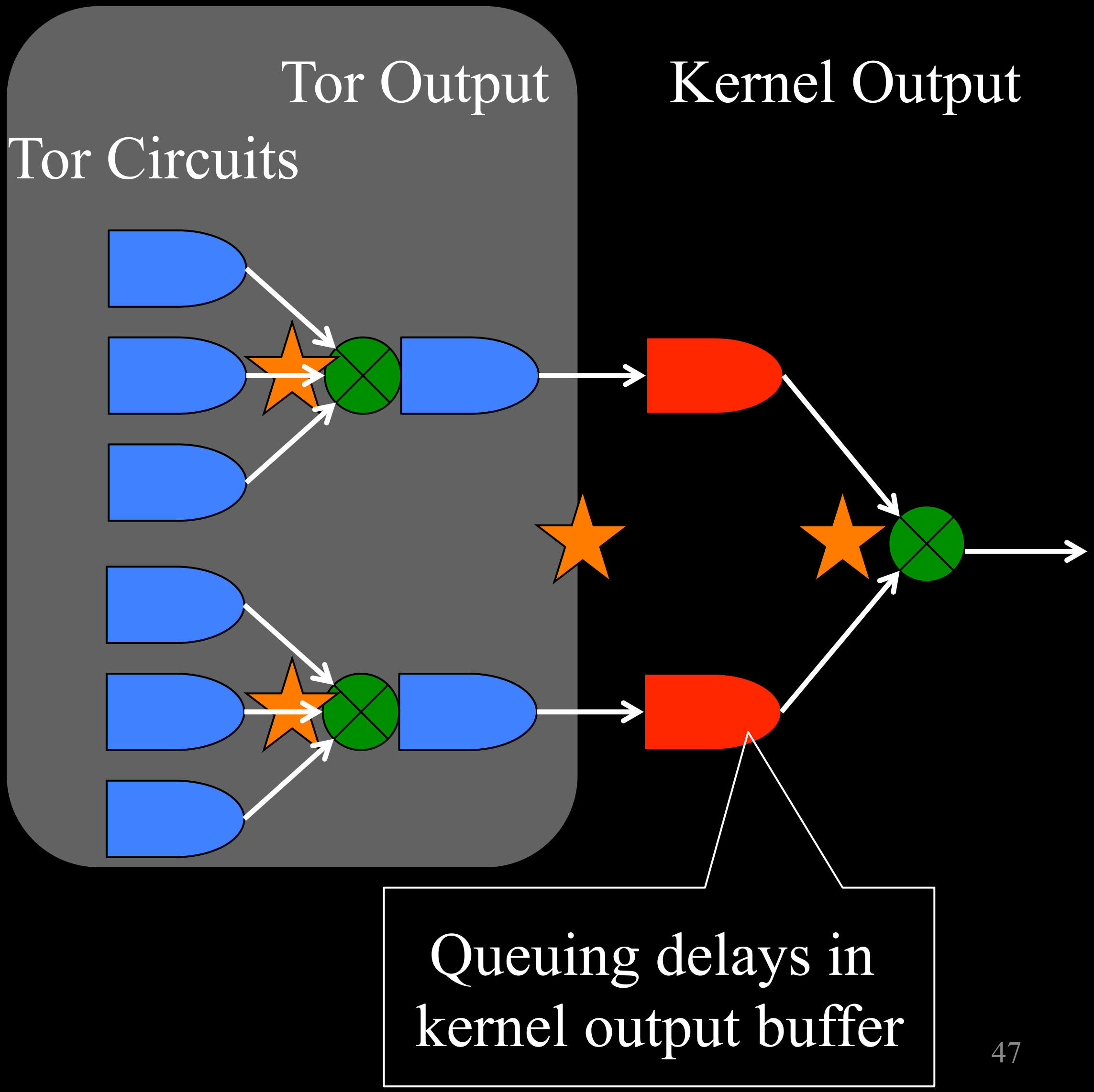


Congestion occurs almost exclusively in outbound kernel buffers

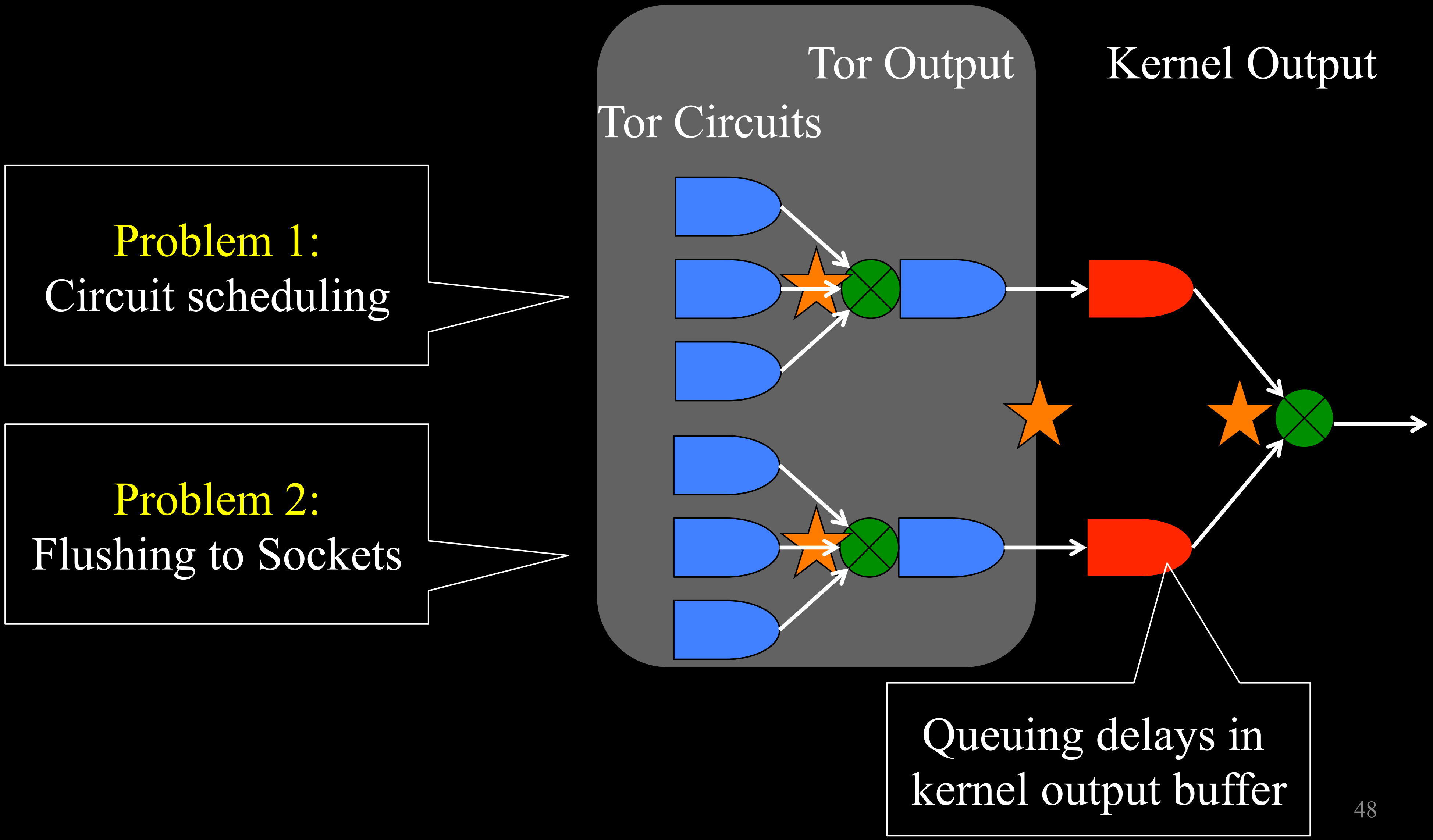
Outline

- ~~Background~~
- ~~Instrument Tor, measure congestion~~
- Analyze causes of congestion
- Design and evaluate KIST
  - Performance
  - Security

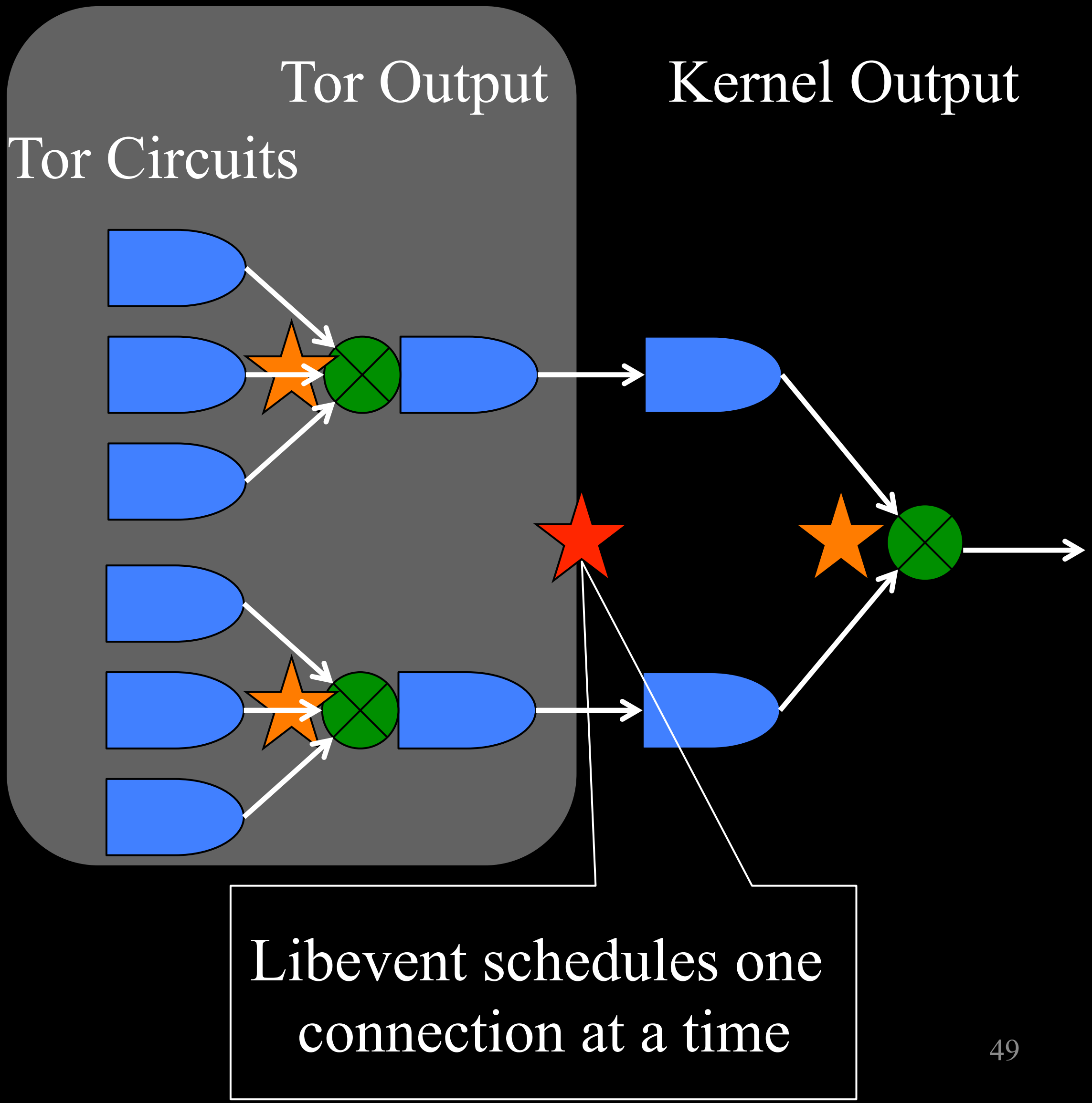
Analyzing Causes of Congestion



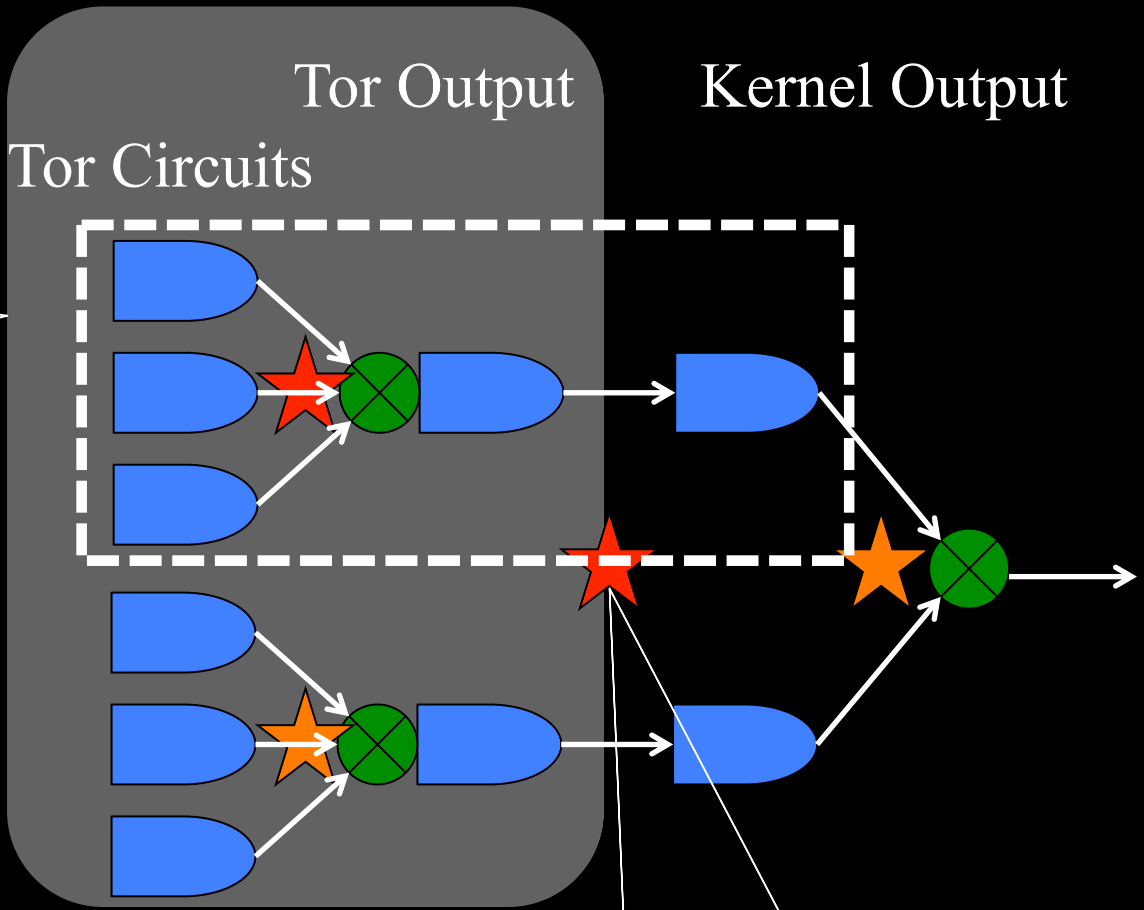
Analyzing Causes of Congestion



Problem 1: Circuit Scheduling



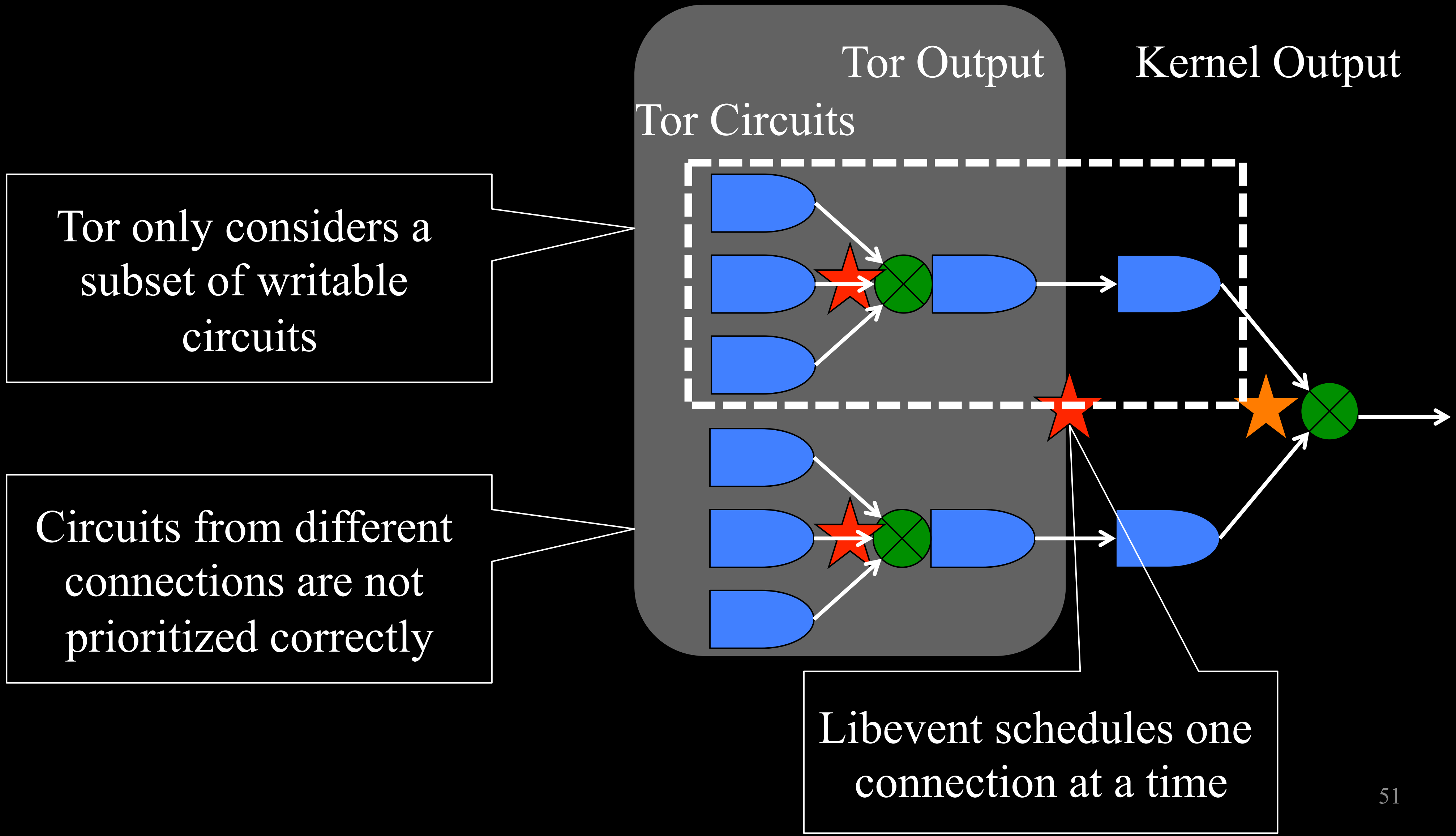
Problem 1: Circuit Scheduling



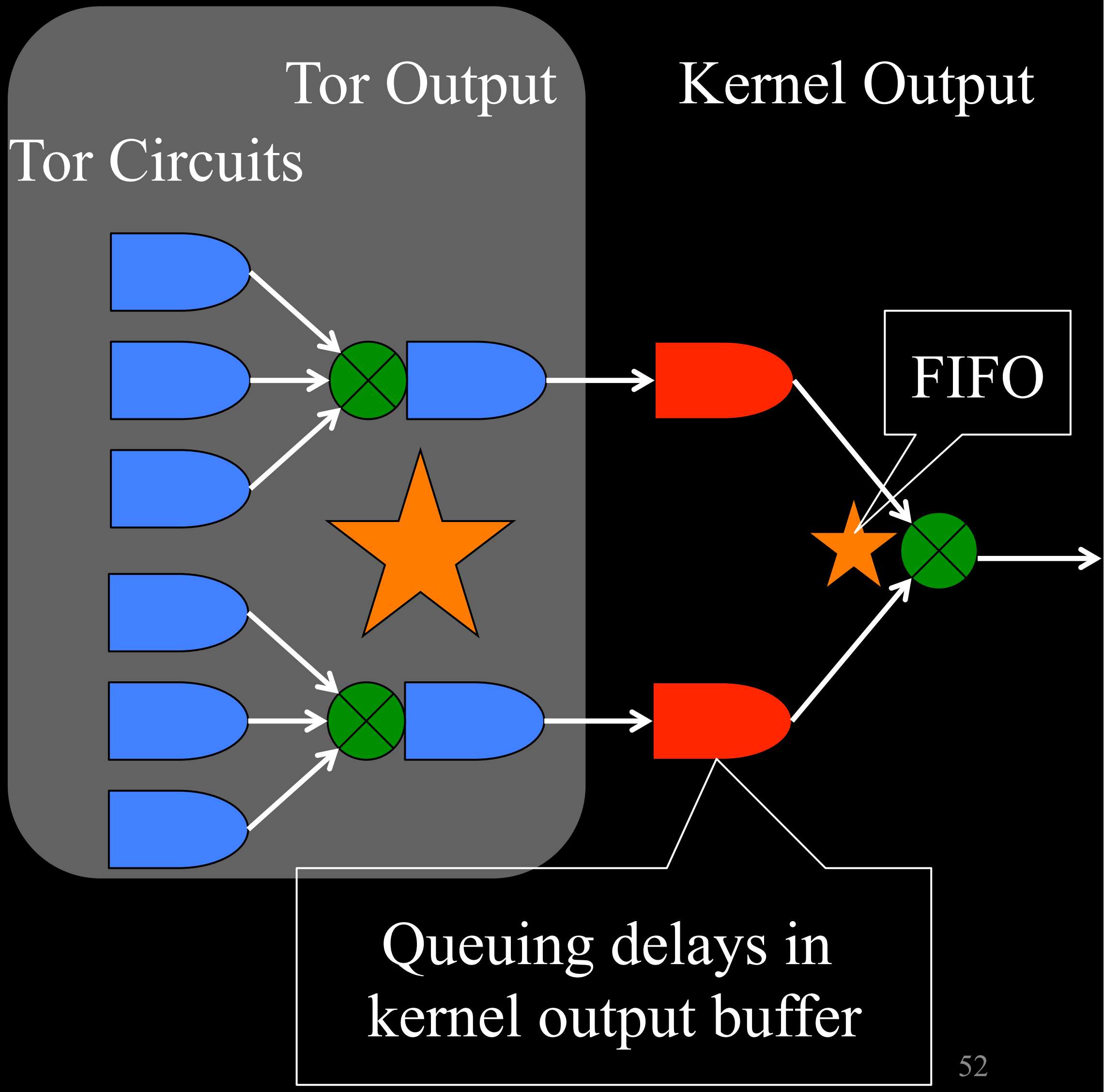
Tor only considers a subset of writable circuits

Libevent schedules one connection at a time

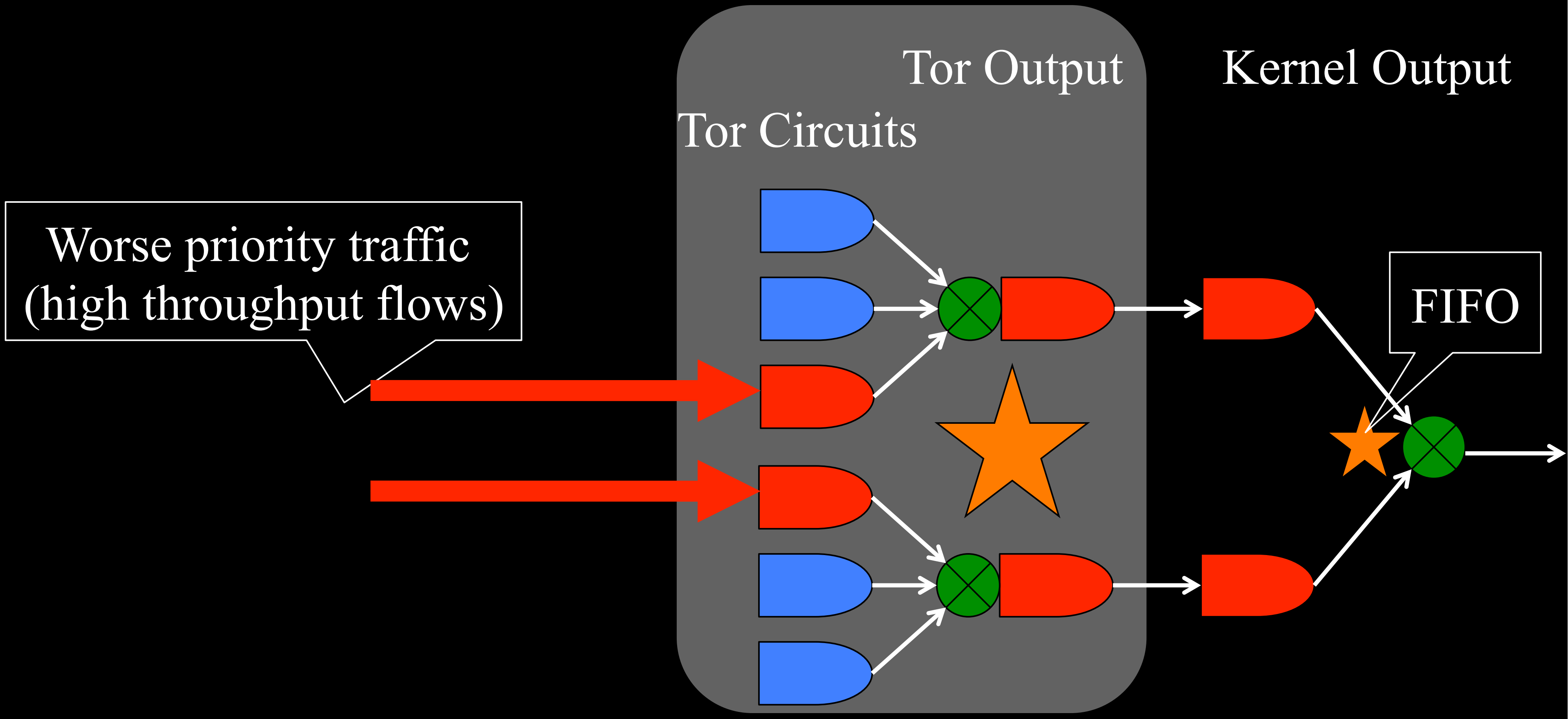
Problem 1: Circuit Scheduling



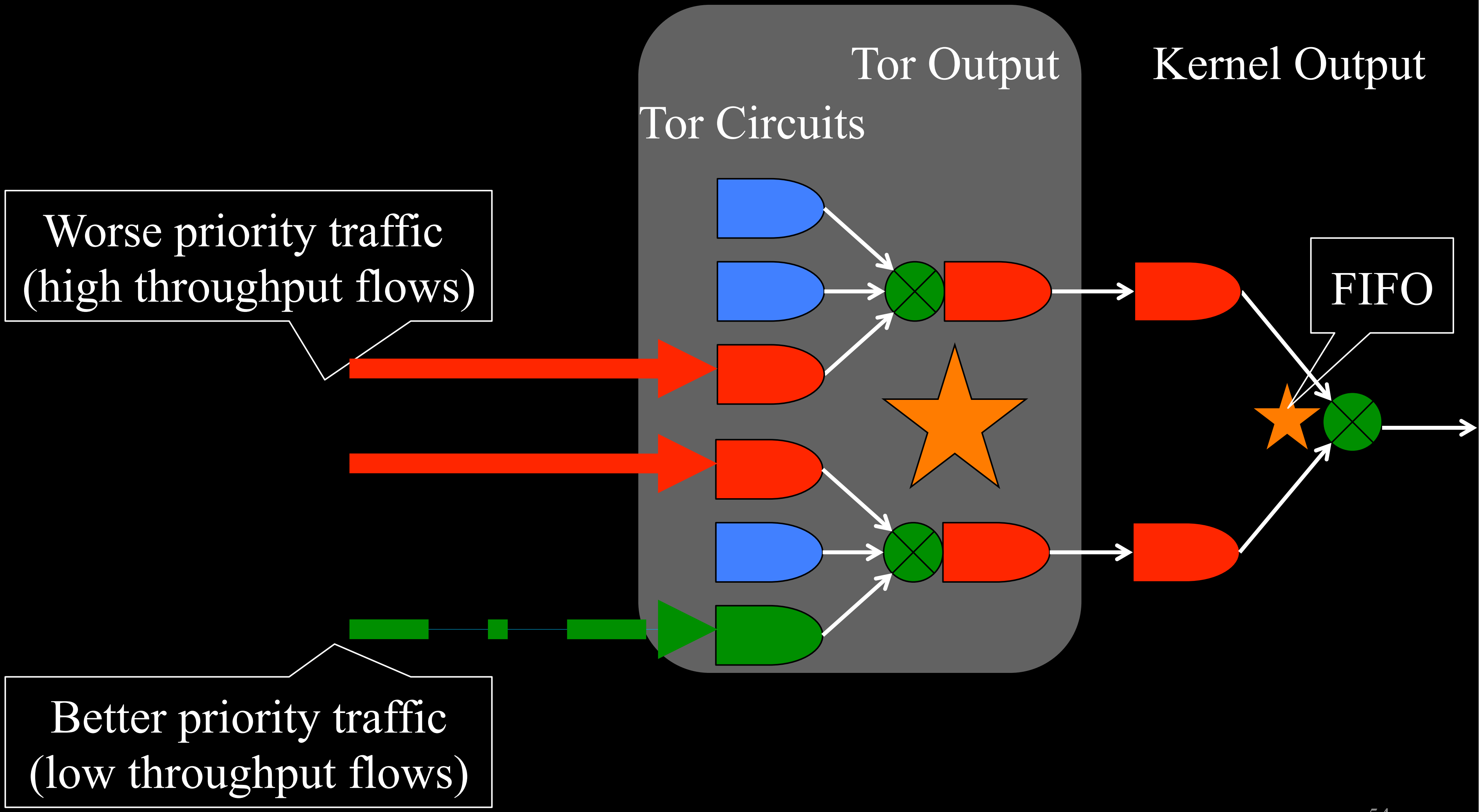
Problem 2: Flushing to Sockets



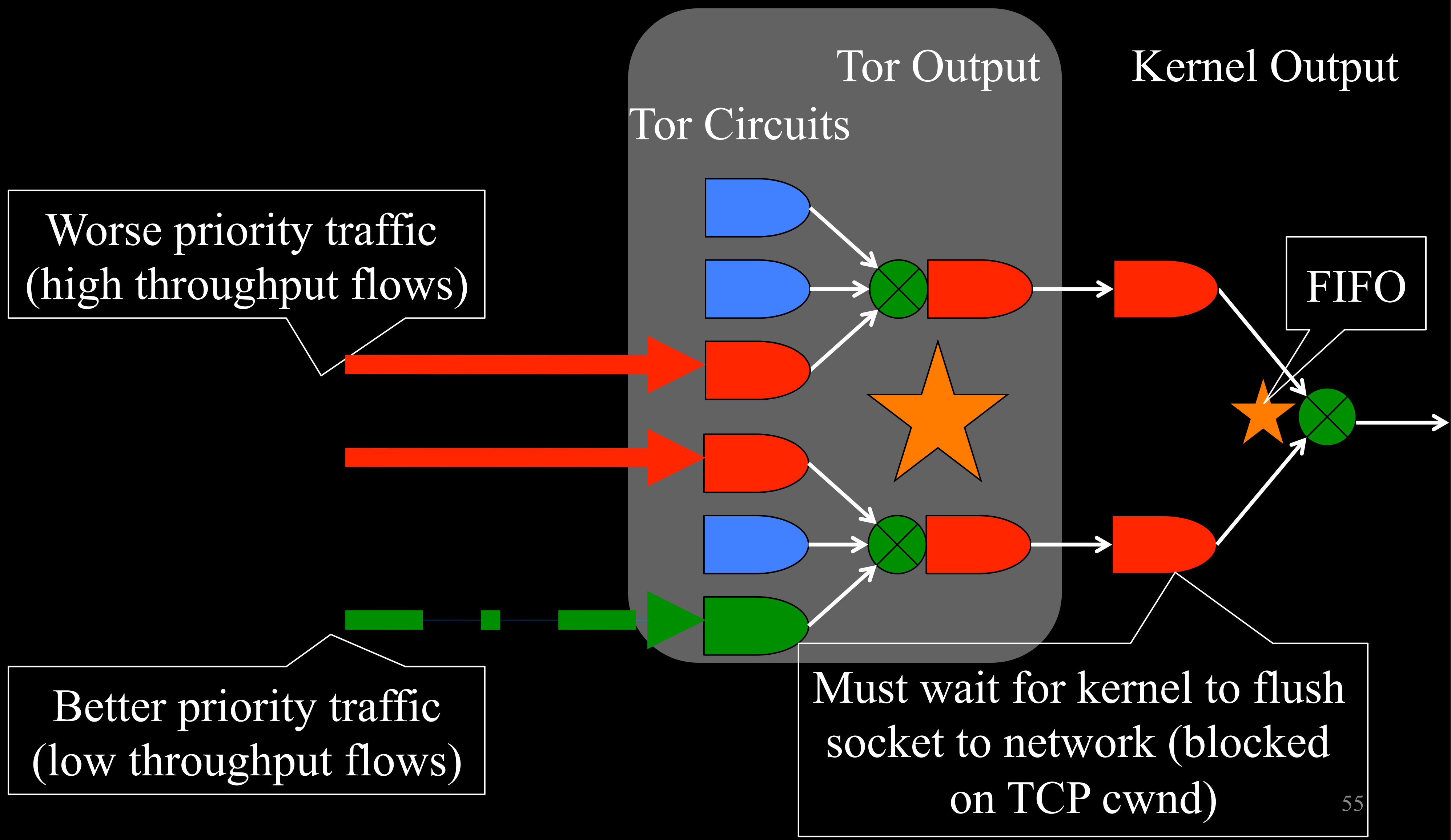
Problem 2: Flushing to Sockets



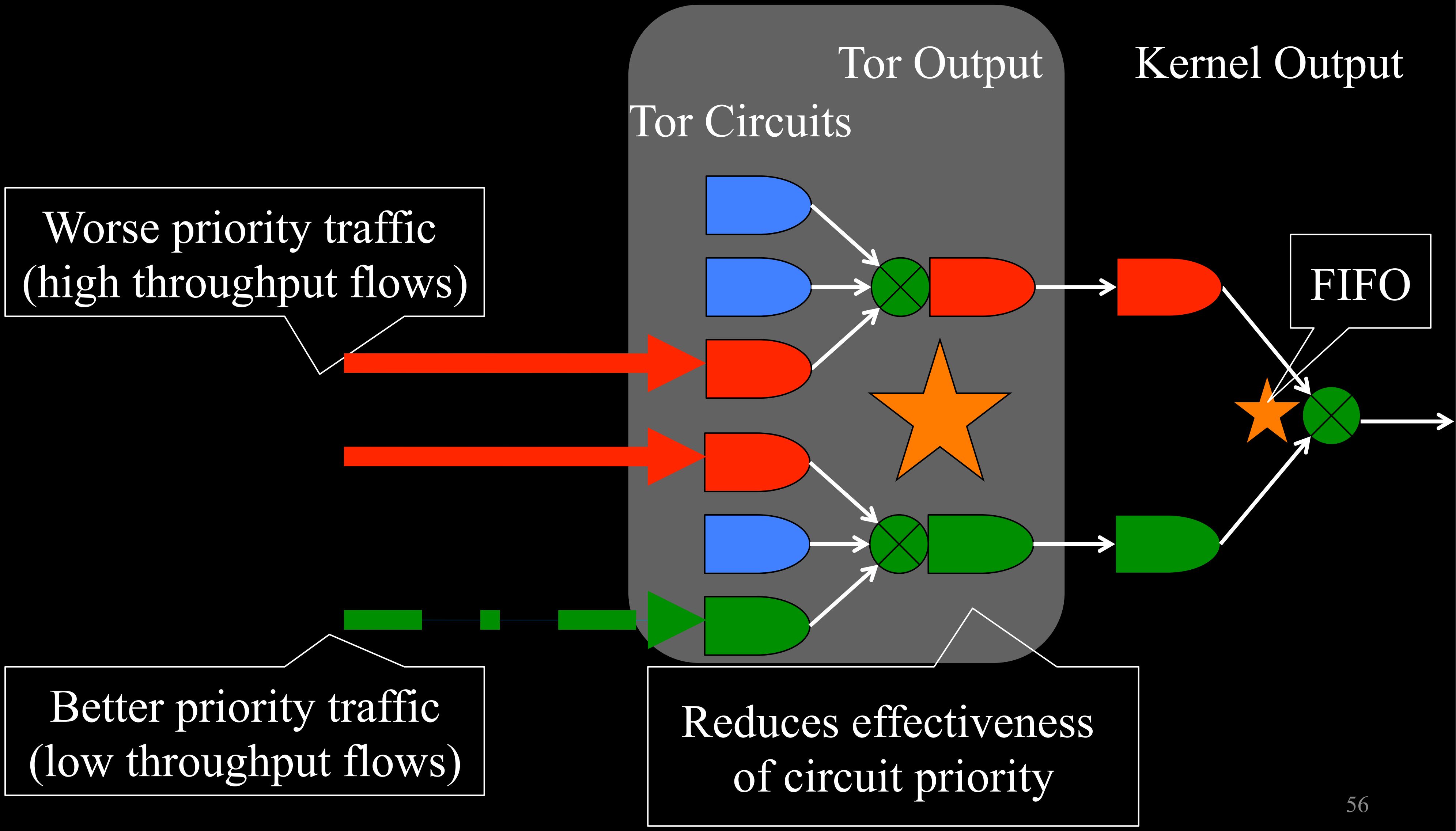
Problem 2: Flushing to Sockets



Problem 2: Flushing to Sockets



Problem 2: Flushing to Sockets



Outline

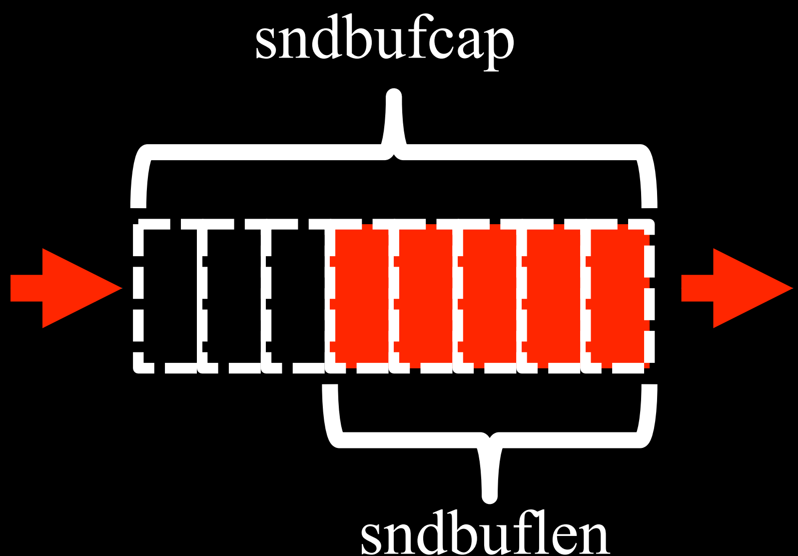
- ~~Background~~
- ~~Instrument Tor, measure congestion~~
- ~~Analyze causes of congestion~~
- Design and evaluate KIST
  - Performance
  - Security

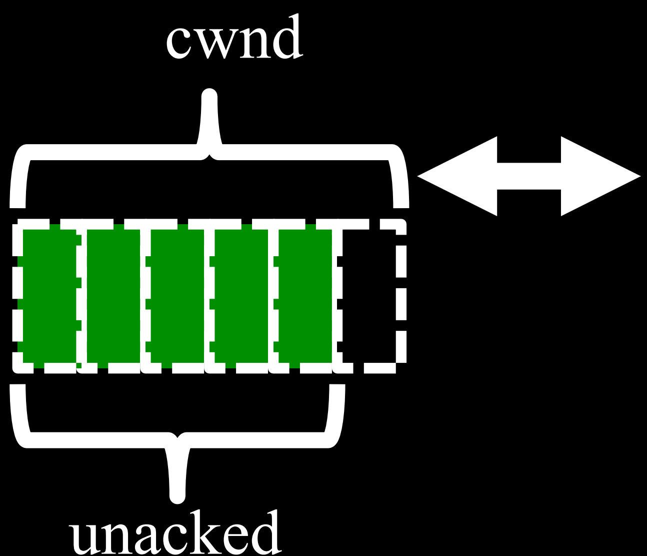
Ask the kernel, stupid!

- Utilize getsockopt and ioctl syscalls

socket_space =
sndbufcap – sndbuflen

tcp_space =
(cwnd – unacked) * mss





Kernel-Informed Socket Transport

- Don't write it if the kernel can't send it; bound kernel writes by:
  - Socket: min(socket_space, tcp_space)
  - Global: upstream bandwidth capacity

Solution to Problem 2

Kernel-Informed Socket Transport

- Don’t write it if the kernel can’t send it; bound kernel writes by:
  - Socket: min(socket_space, tcp_space)
  - Global: upstream bandwidth capacity
- Choose globally from all writable circuits

Solution to Problem 1

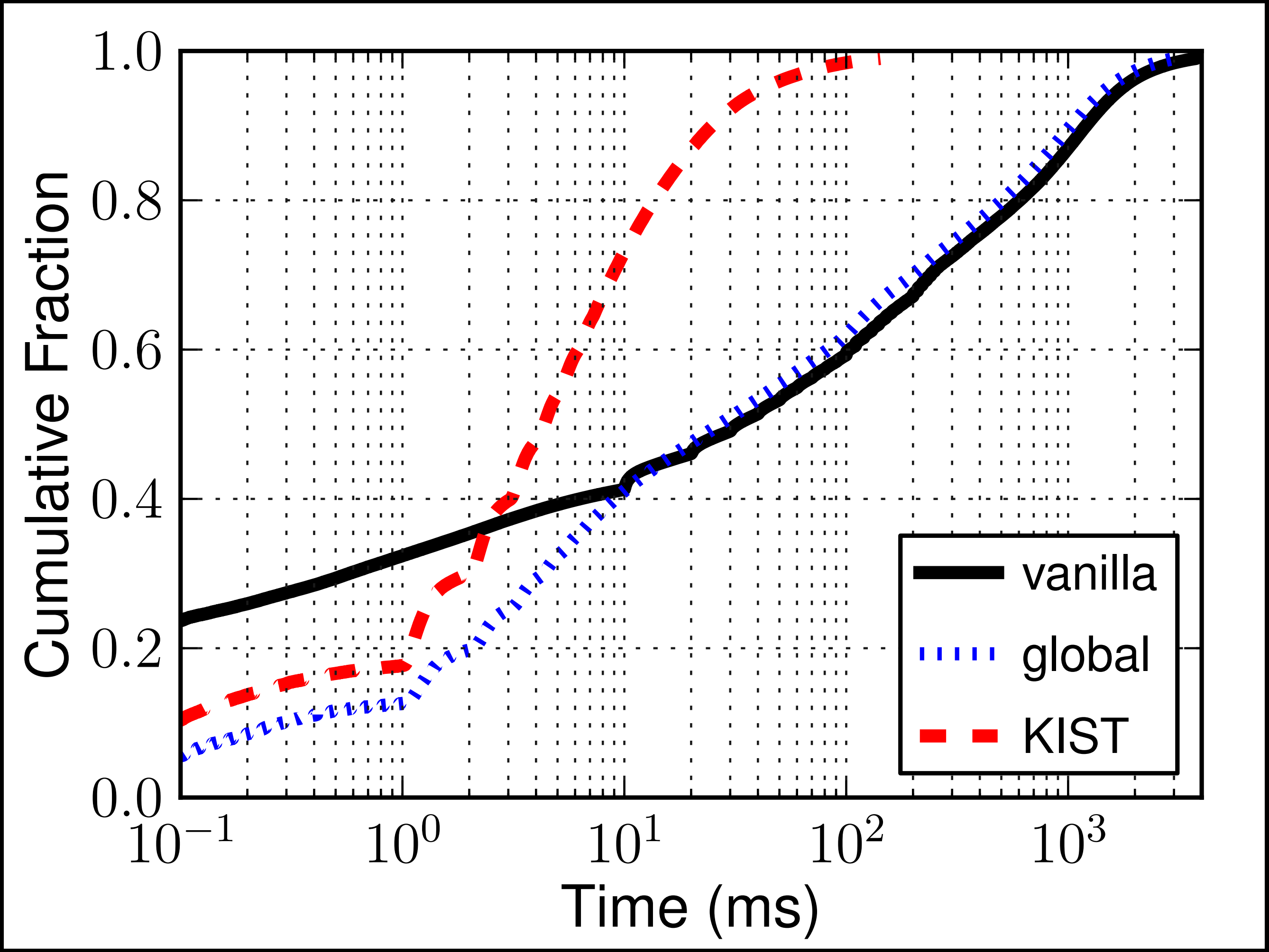
Kernel-Informed Socket Transport

- Don't write it if the kernel can't send it; bound kernel writes by:
  - Socket: min(socket_space, tcp_space)
  - Global: upstream bandwidth capacity

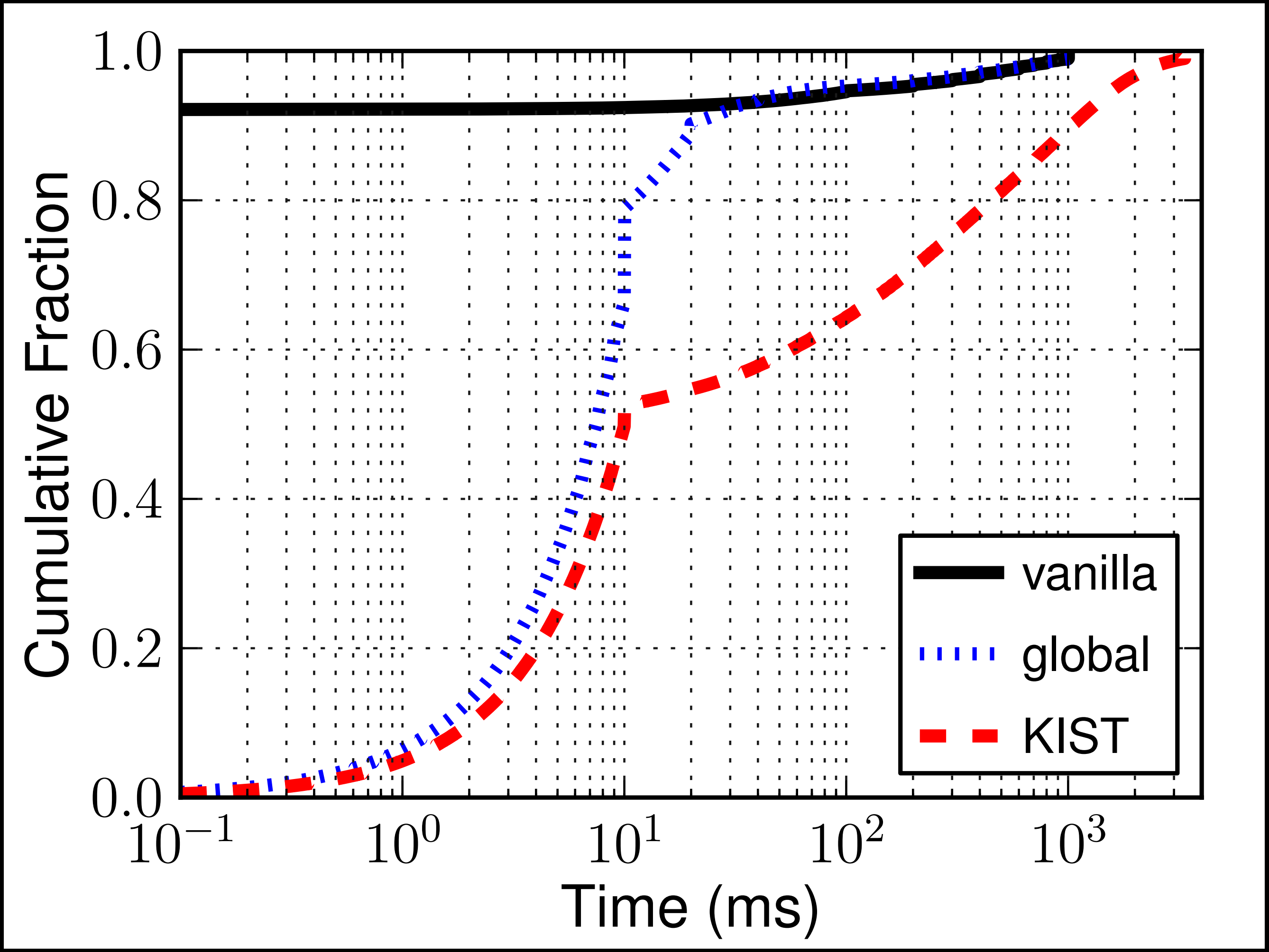
- Choose globally from all writable circuits

- Try to write again before kernel starvation

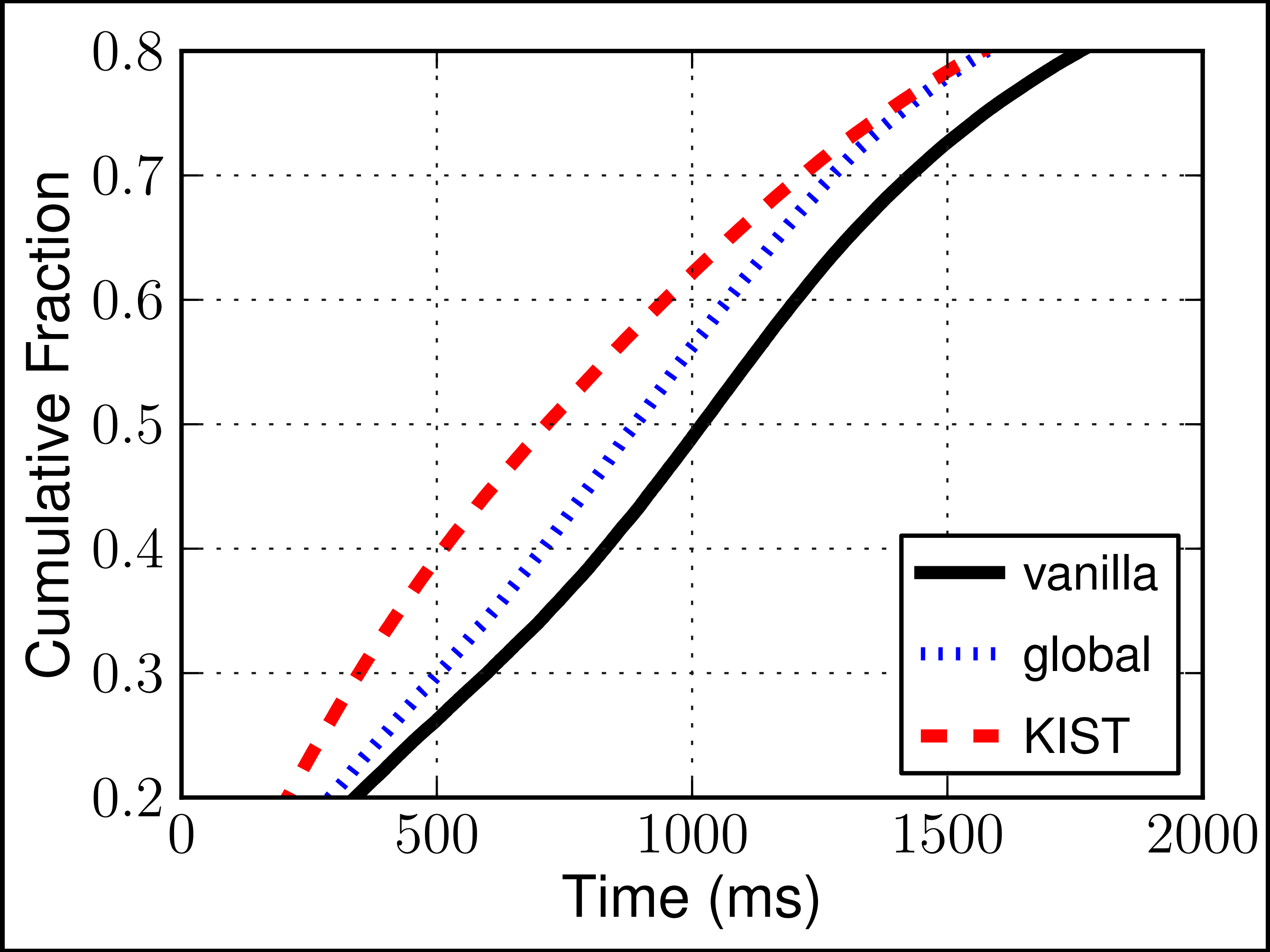
KIST Reduces Kernel Congestion



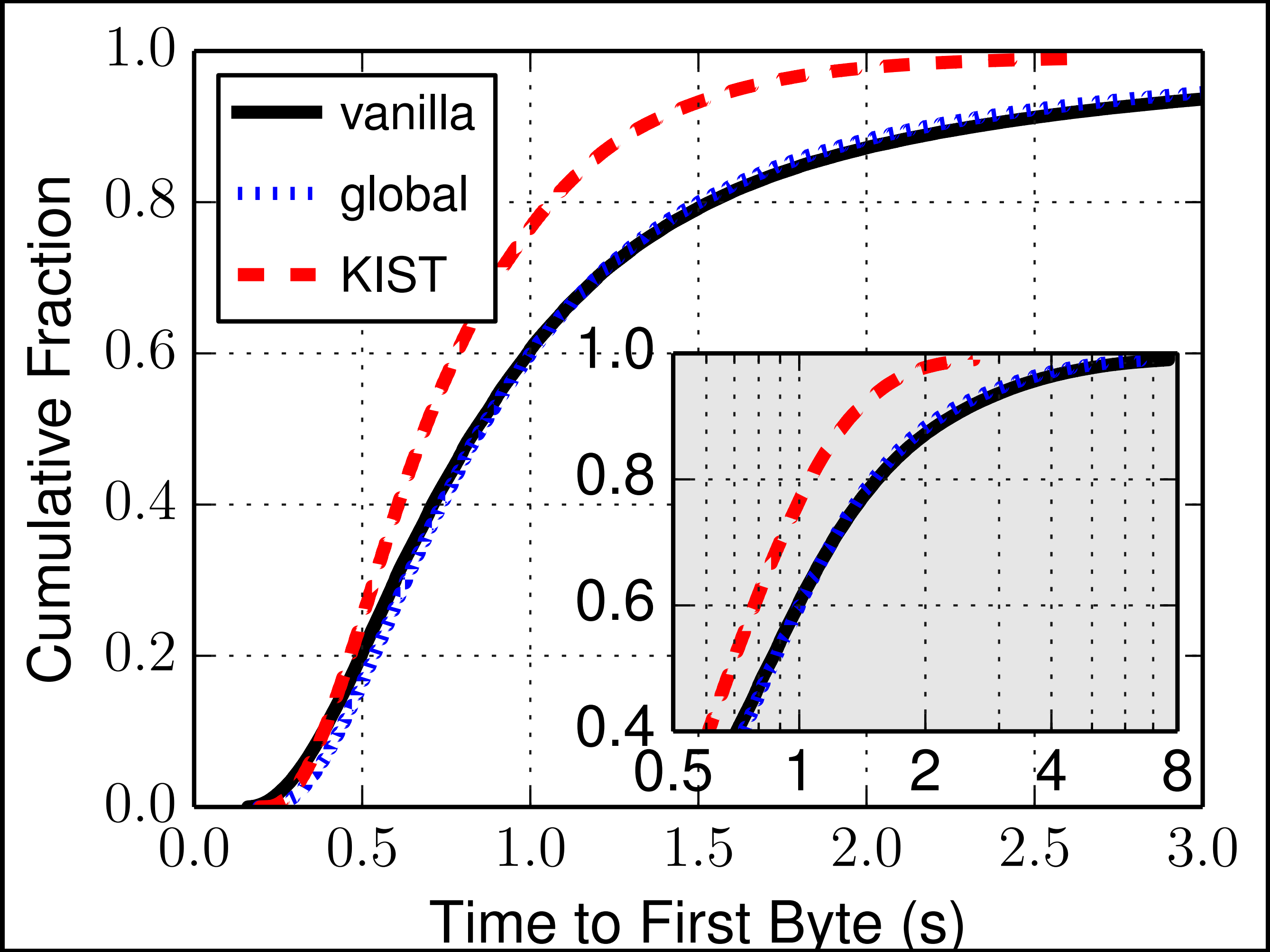
KIST Increases Tor Congestion



KIST Reduces Circuit Congestion



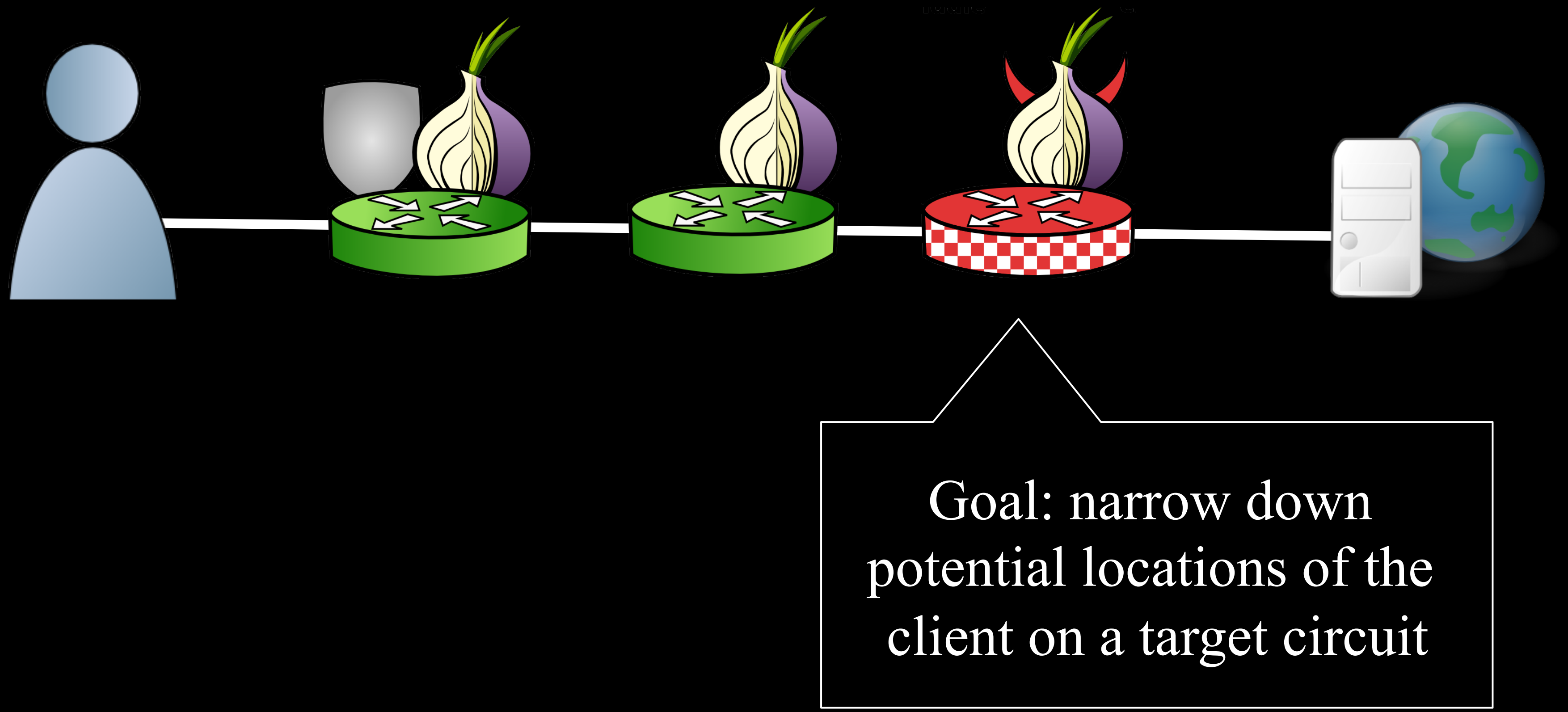
KIST Improves Network Latency



Outline

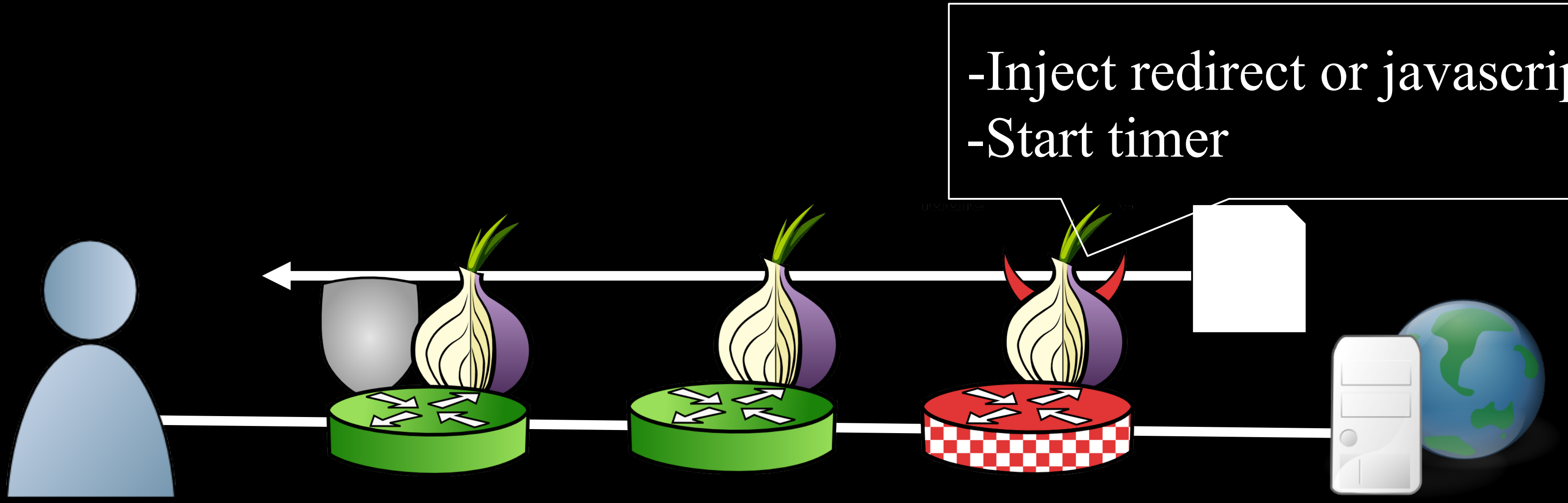
- ~~Background~~
- ~~Instrument Tor, measure congestion~~
- ~~Analyze causes of congestion~~
- Design and evaluate KIST
  - ~~Performance~~
  - Security

Traffic Correlation: Latency



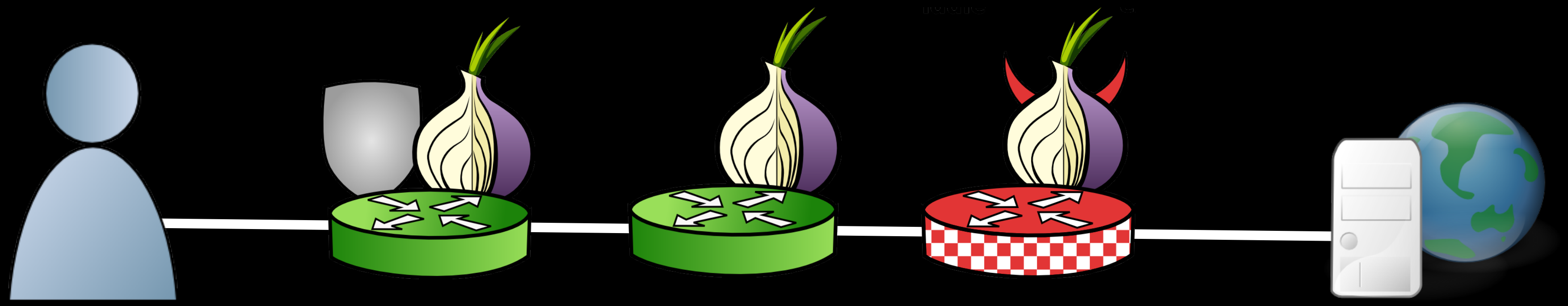
Hopper et.al. CCS'07

Traffic Correlation: Latency



Hopper et.al. CCS'07

Traffic Correlation: Latency

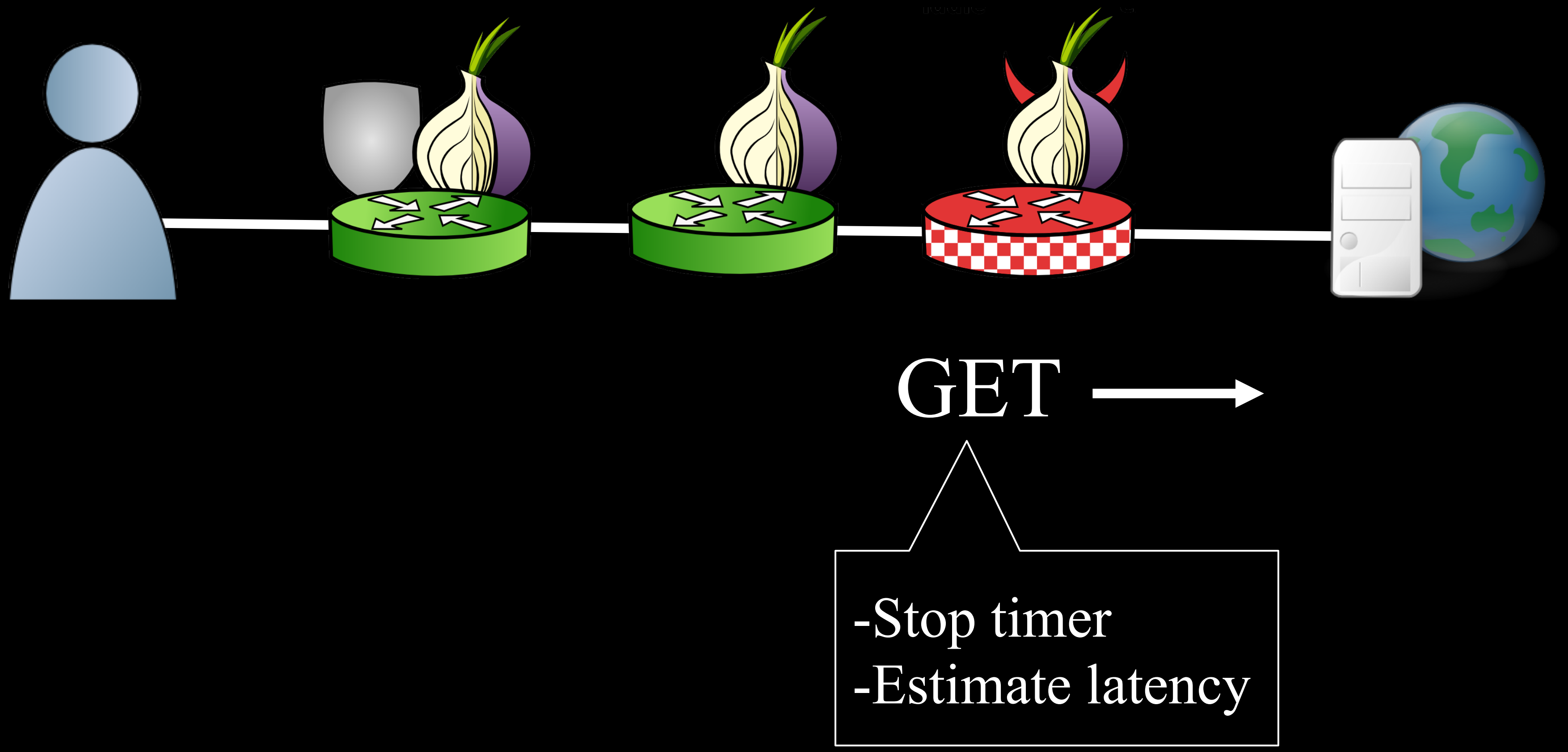


GET

Request redirected page or embedded object

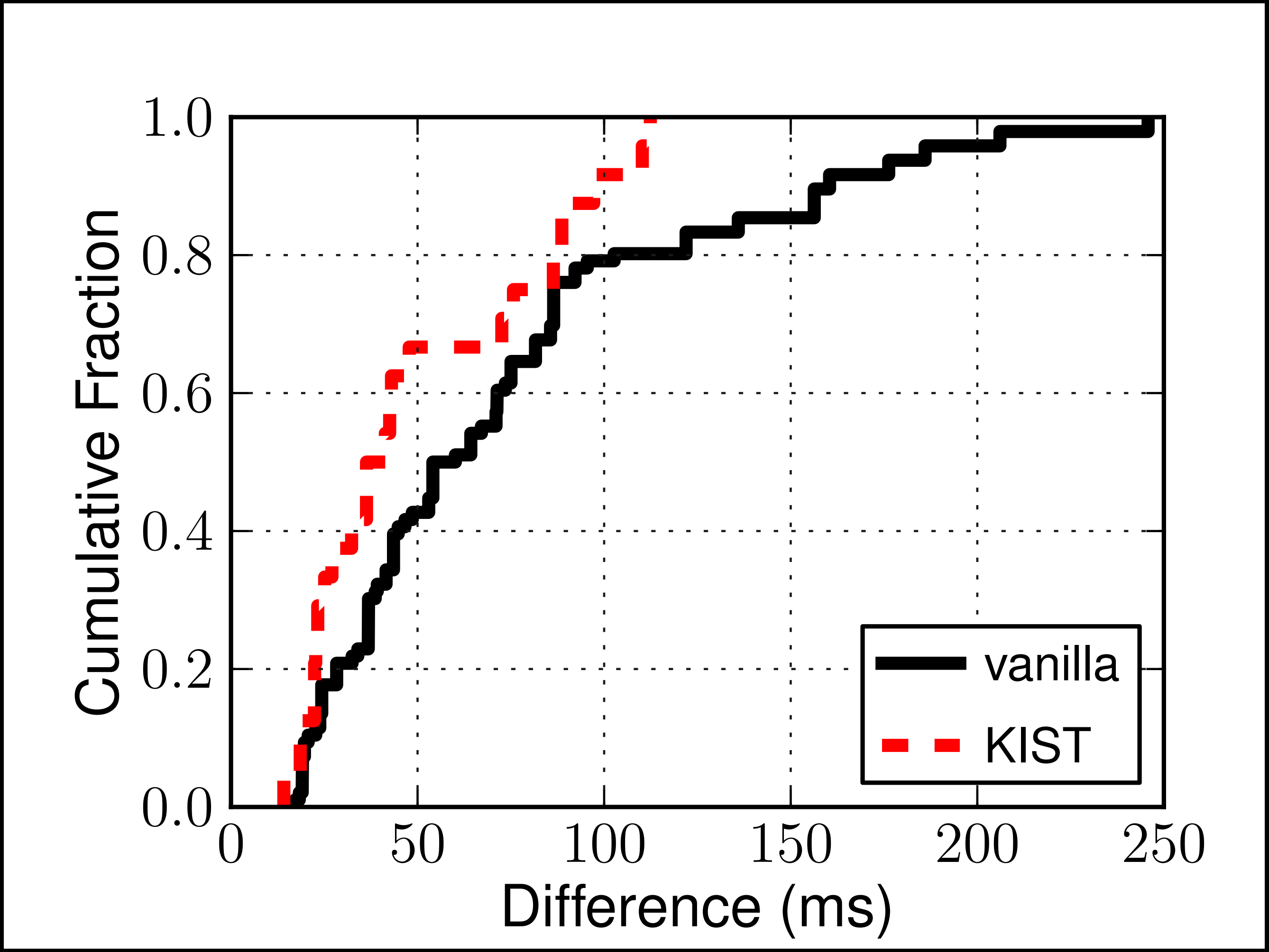
Hopper et.al. CCS'07

Traffic Correlation: Latency

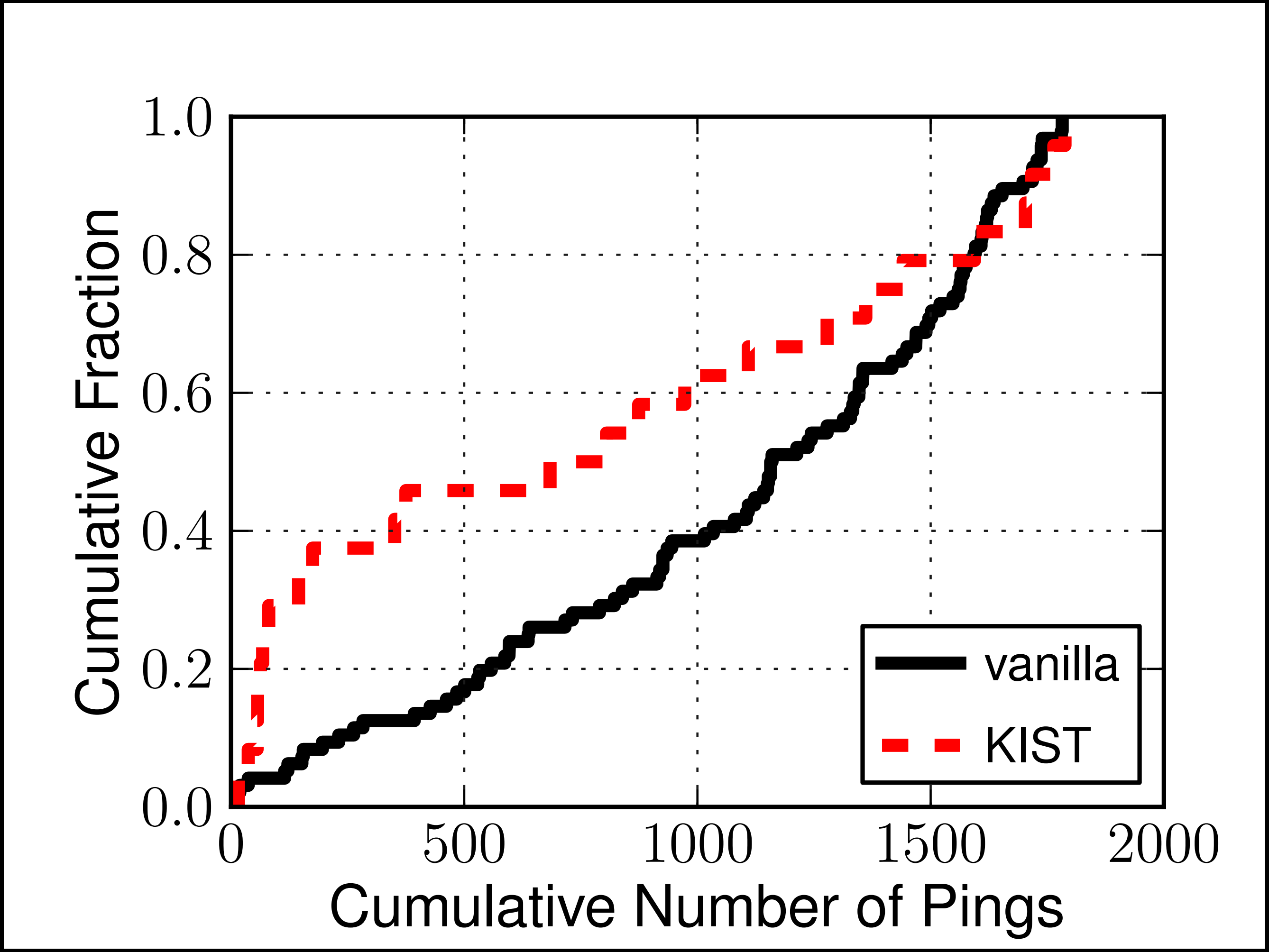


Hopper et.al. CCS'07

Latency Attack
| estimate – actual |



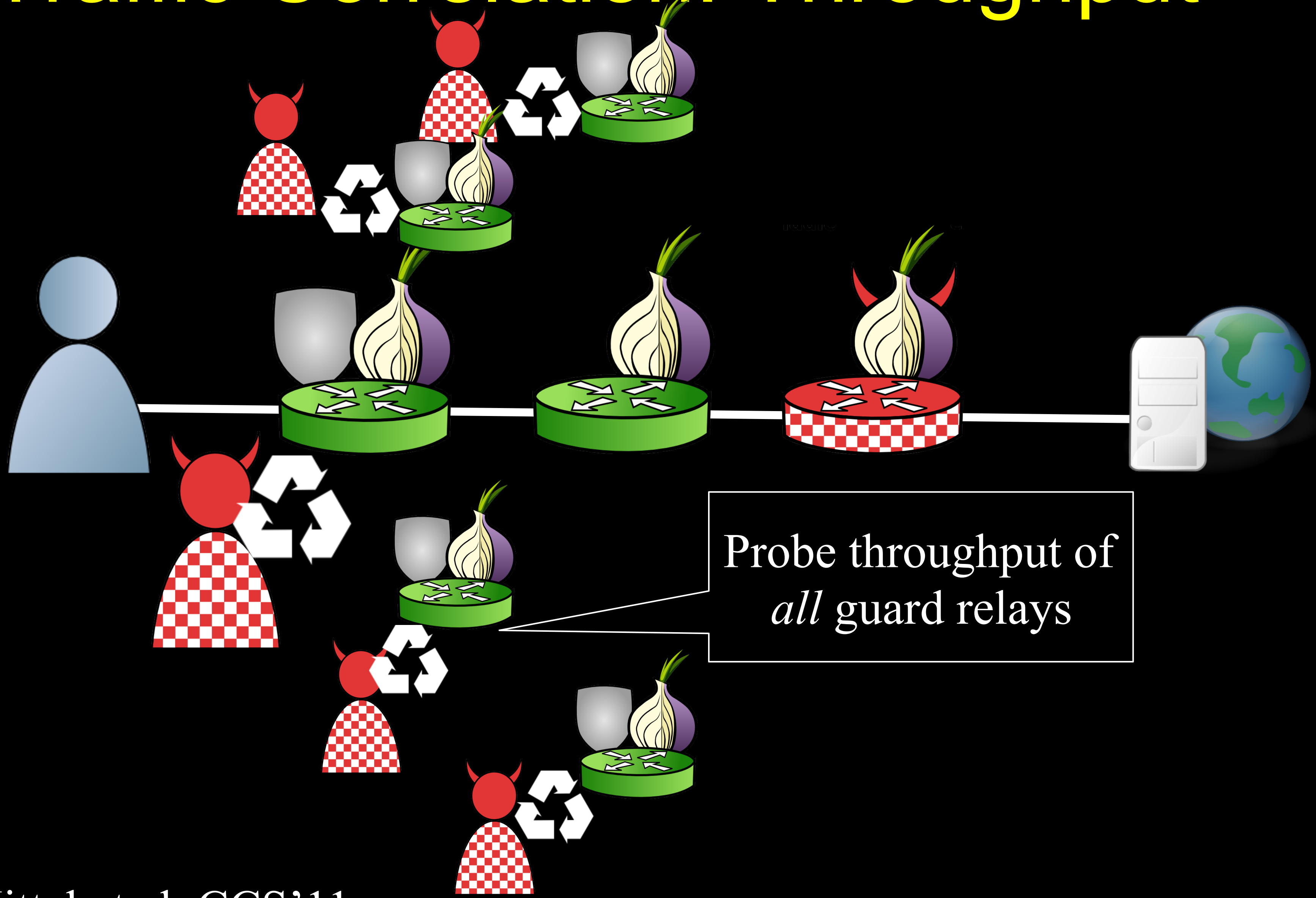
Latency Attack
num pings until best estimate



Traffic Correlation: Throughput

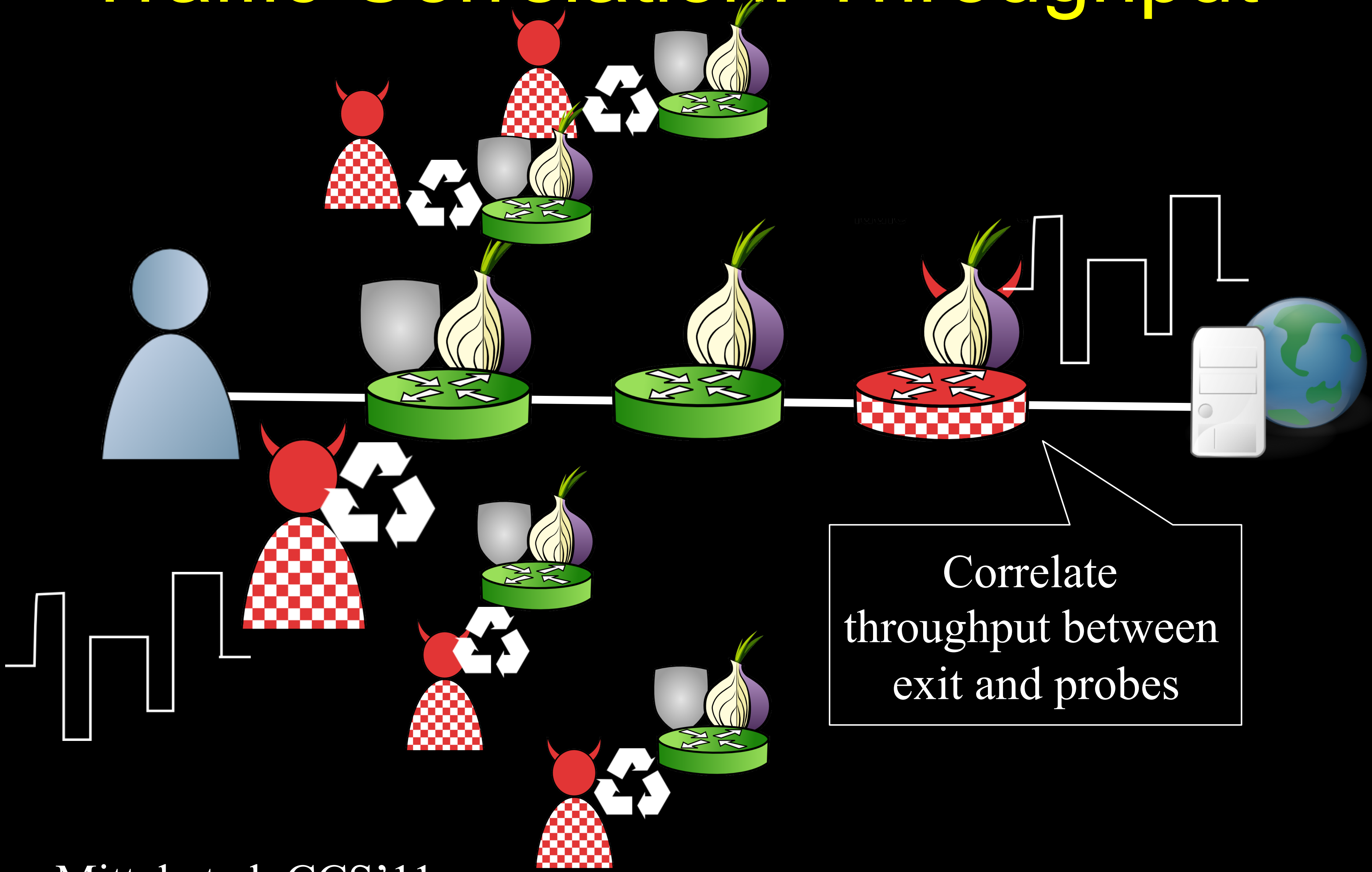
Mittal et.al. CCS'11

Traffic Correlation: Throughput



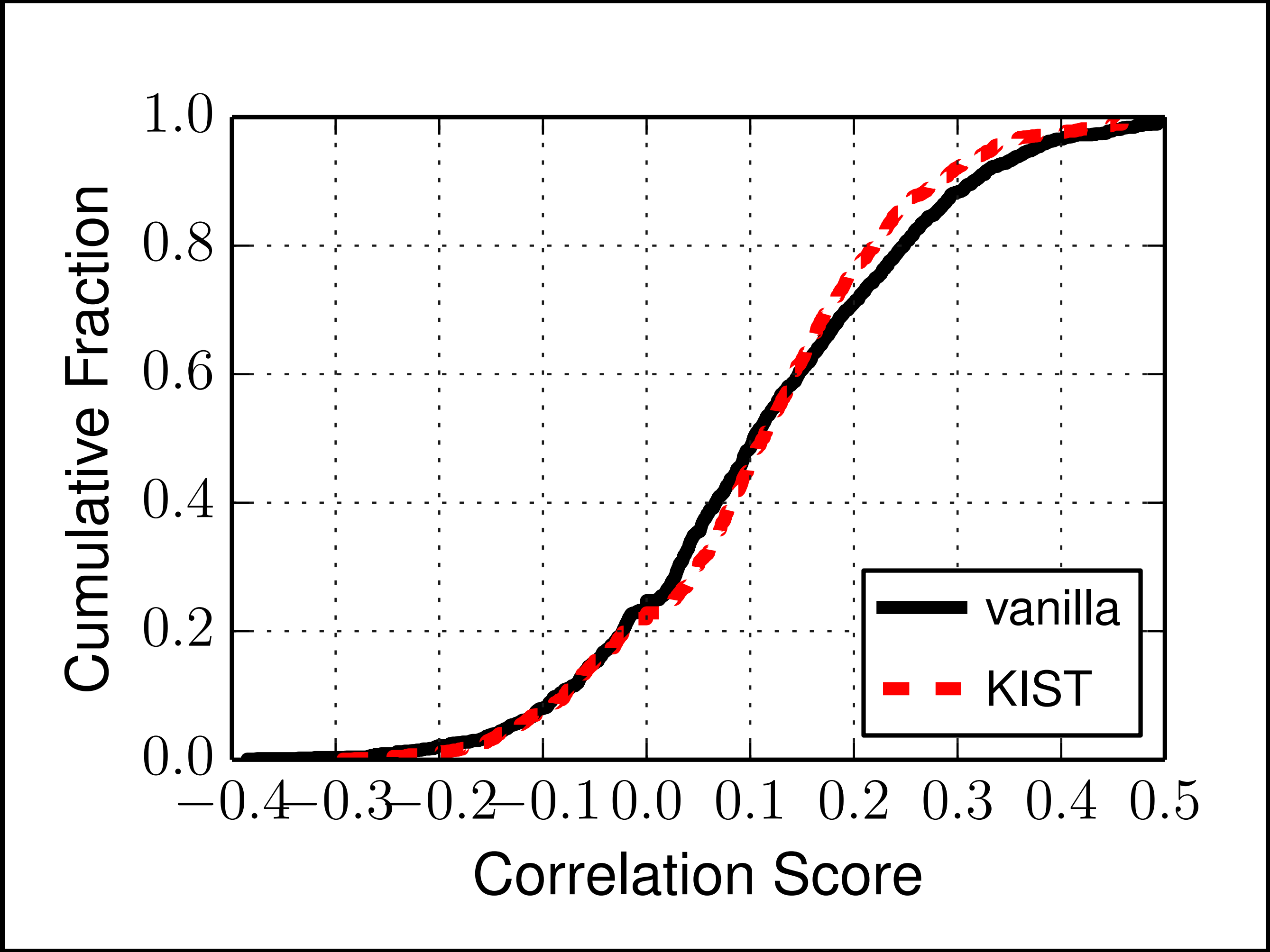
Mittal et.al. CCS'11

Traffic Correlation: Throughput



Mittal et.al. CCS'11

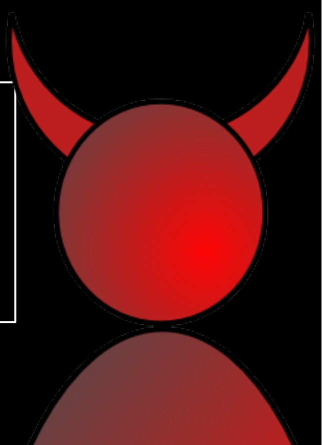
Throughput Attack Results



Summary/Conclusion

- Shadow
- Where is Tor slow?
  - KIST complements other performance enhancements, e.g. circuit priority
- Future work
  - Optimize Shadow threading algorithms
  - Distribute Shadow across processes/machines

shadow.github.io github.com/shadow	robgjansen.com, @robgjansen rob.g.jansen@nrl.navy.mil



think like an adversary