Unified Debugging of Distributed Systems with Recon

Kyu Hyung Lee, Nick Sumner,
Xiangyu Zhang, Patrick Eugster

DSN 2011, June 27 - 30

PURDUE
UNIVERSITY

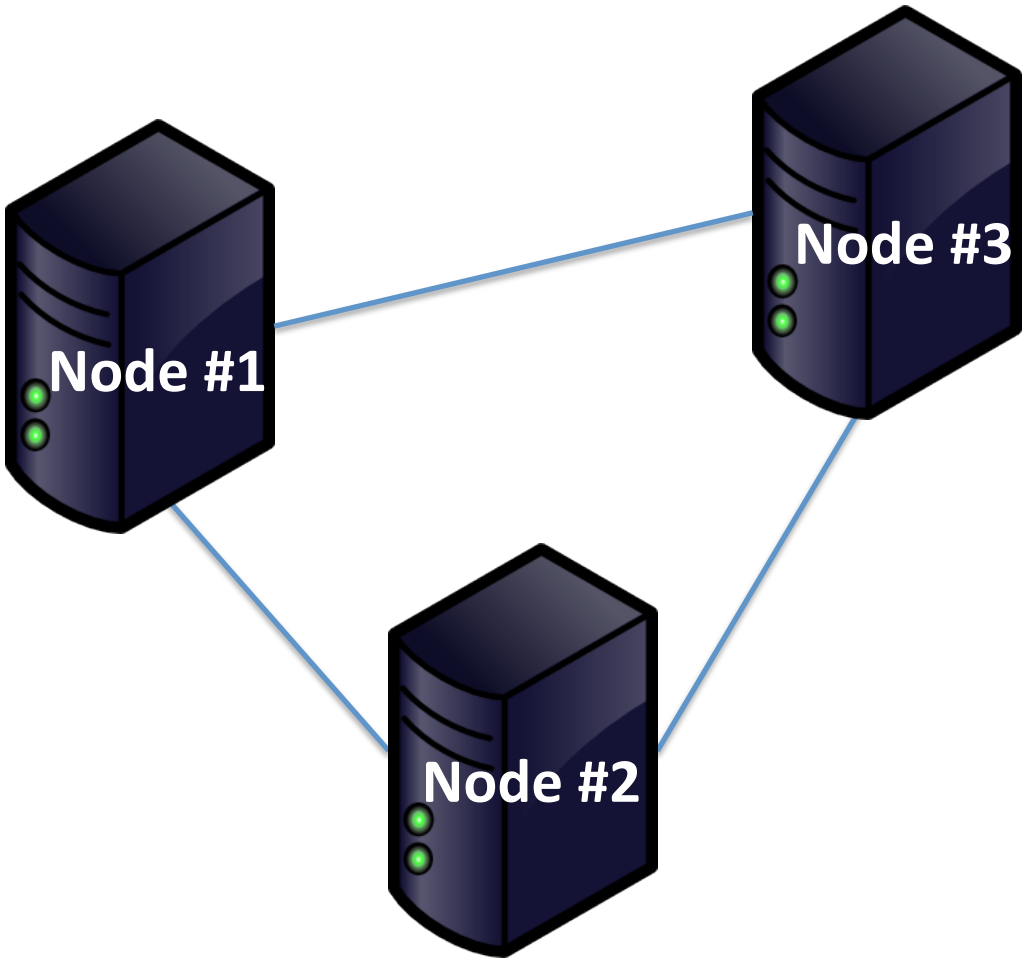
Introduction

- Debugging distributed system is challenging
  - Developers only see a local view
  - The root cause may be in a large distance from the user view

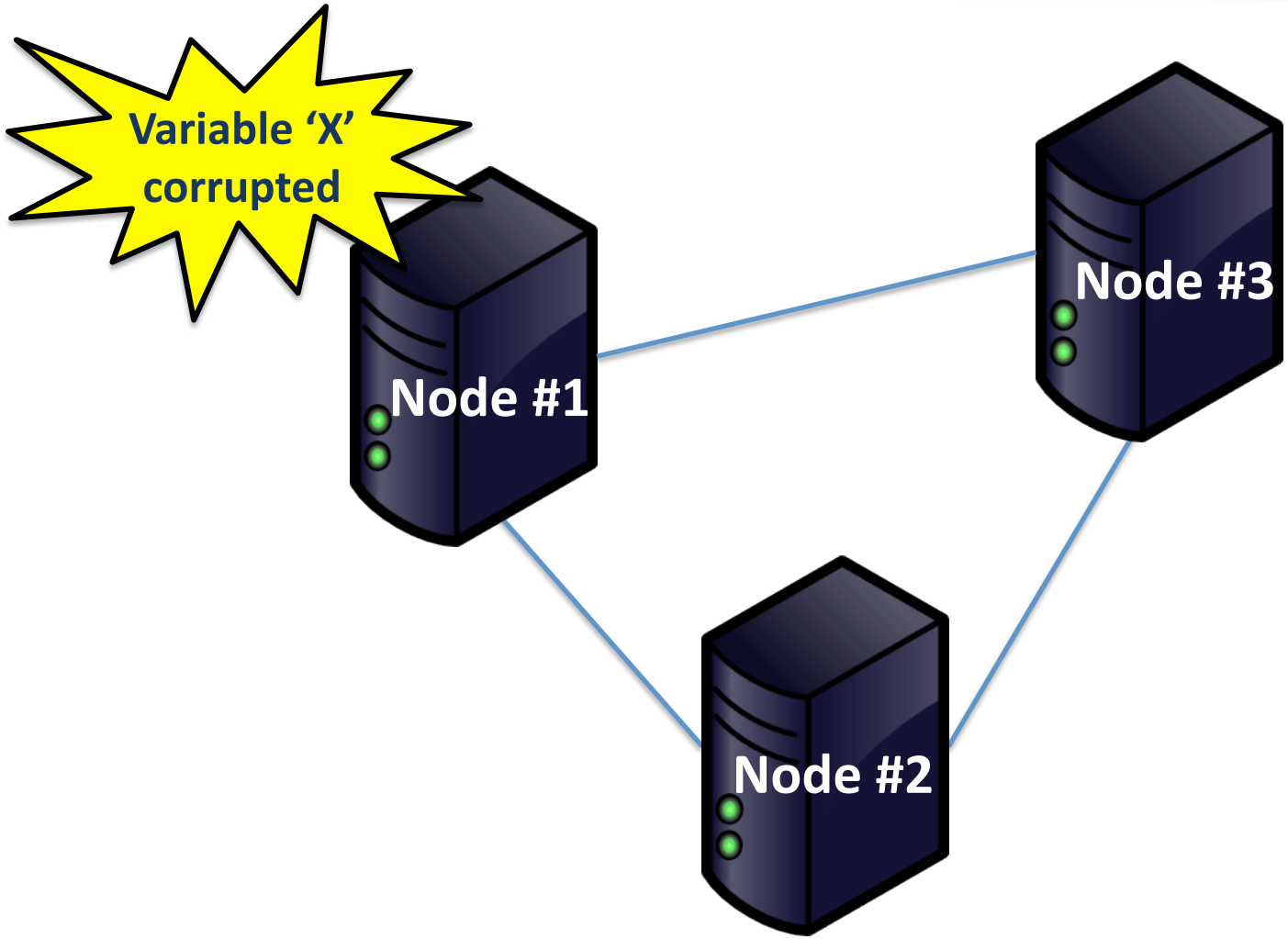
Introduction

- Debugging distributed system is challenging
  - Developers only see a local view
  - Maybe the root cause is in large distance from the user view
- Recording and replay technique
  - Inspect step-wise
  - Need to combine techniques in different levels
    - Instruction level tracing, Event log analysis, Global state reconstruction

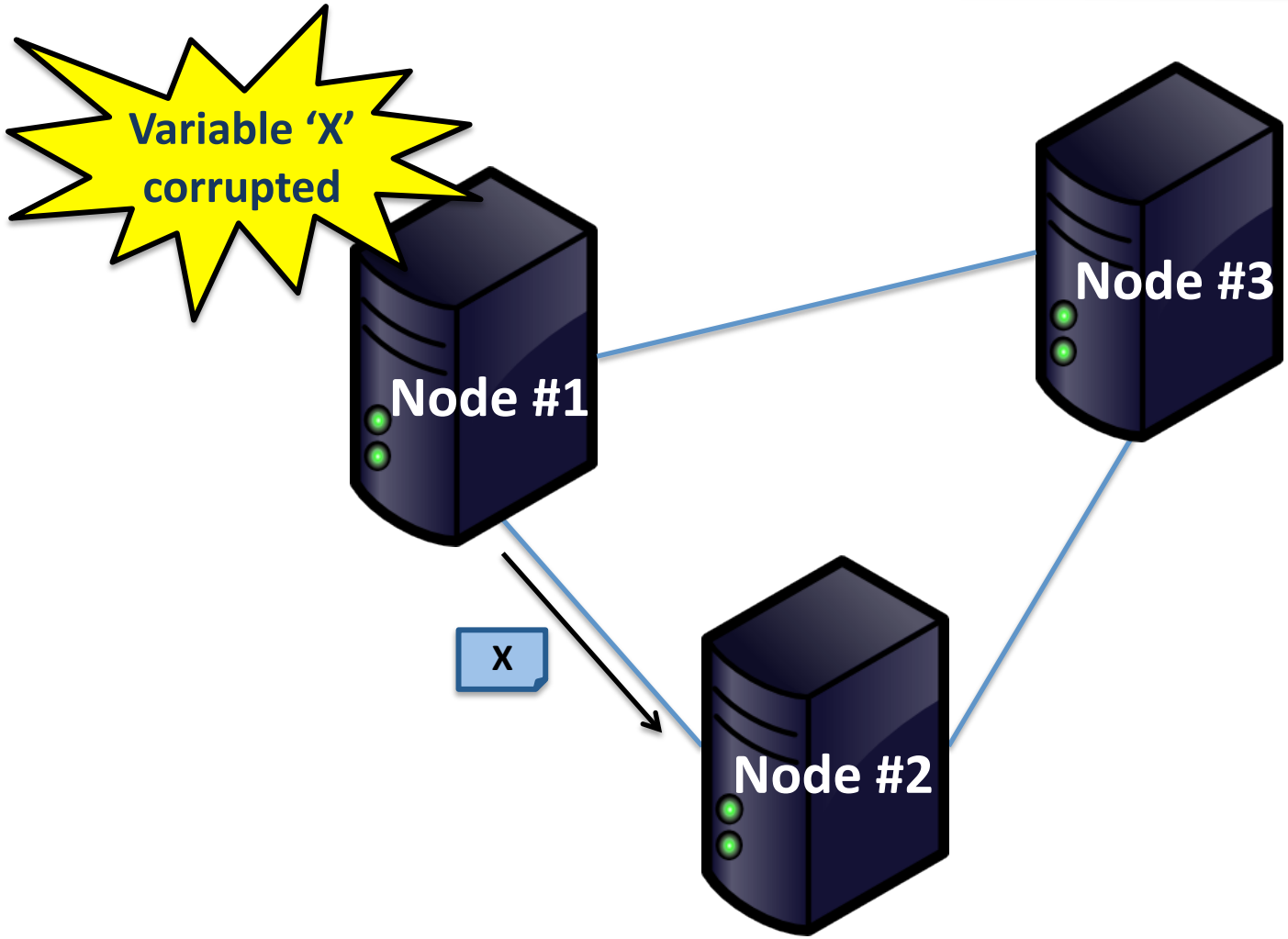
Introduction



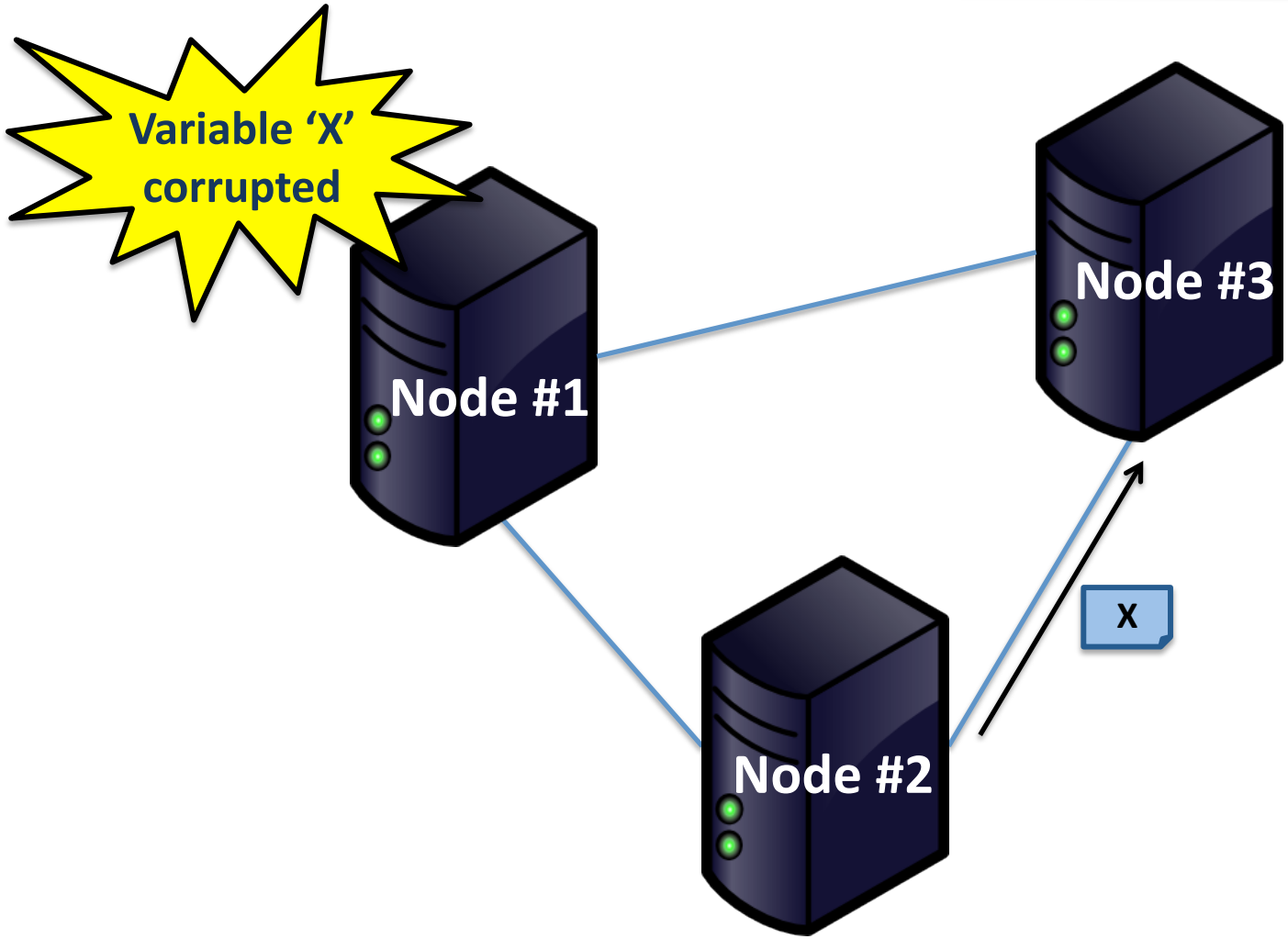
Introduction



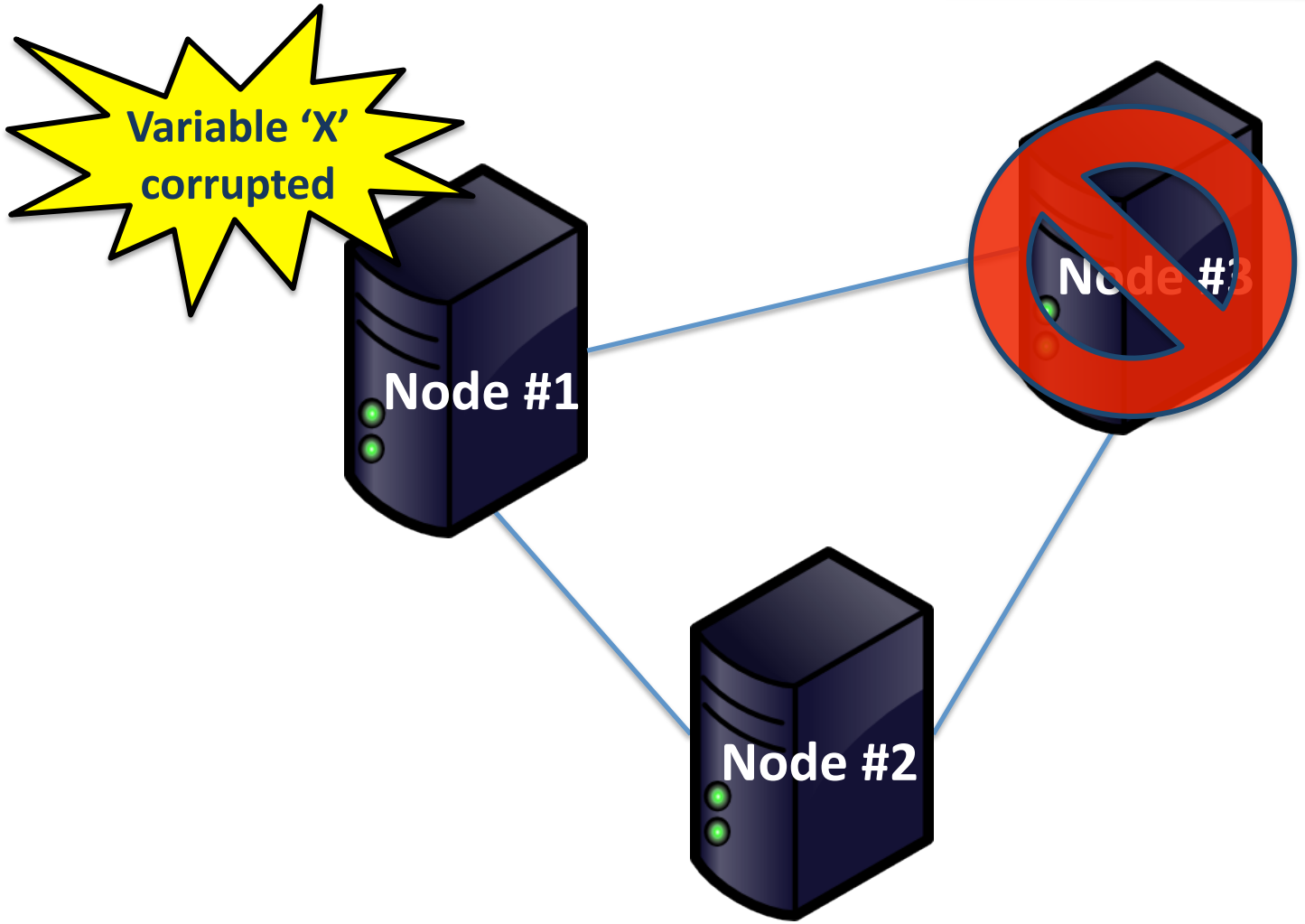
Introduction



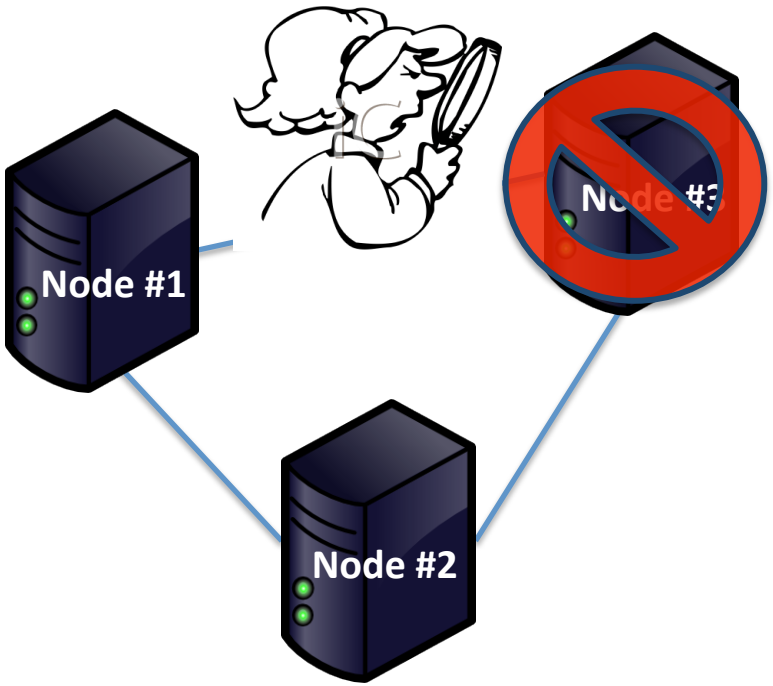
Introduction



Introduction

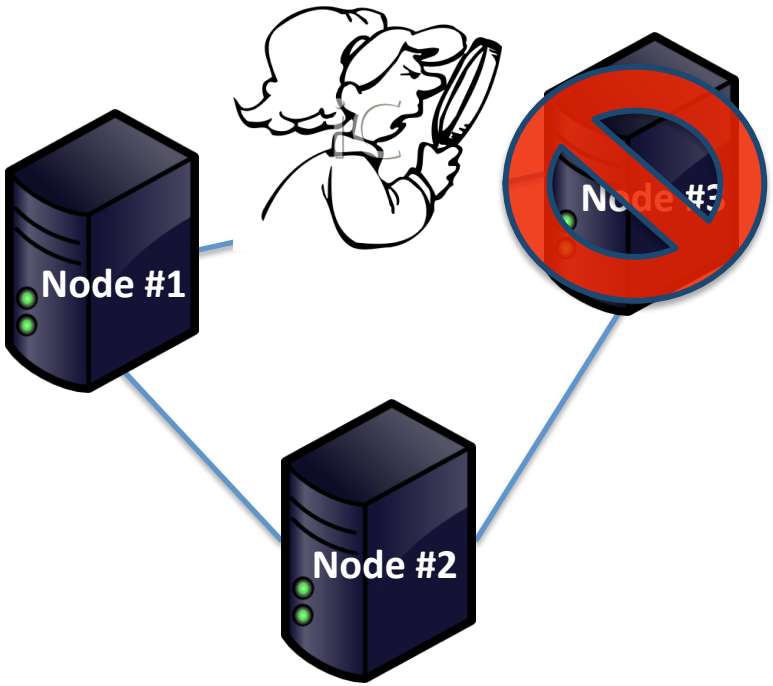


Introduction



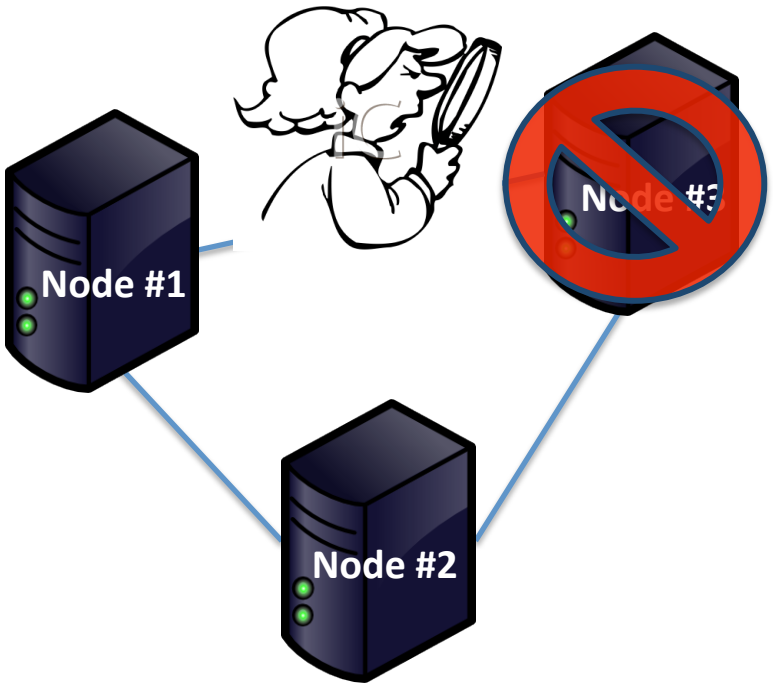
- Where is the program location that ‘x’ defined?

Introduction



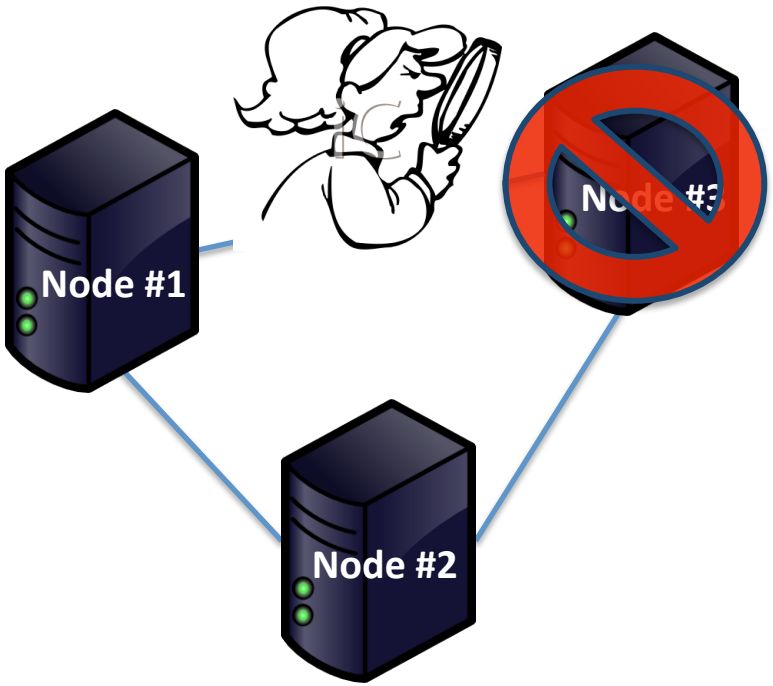
- Where is the program location that ‘x’ defined?
  -> From network..

Introduction



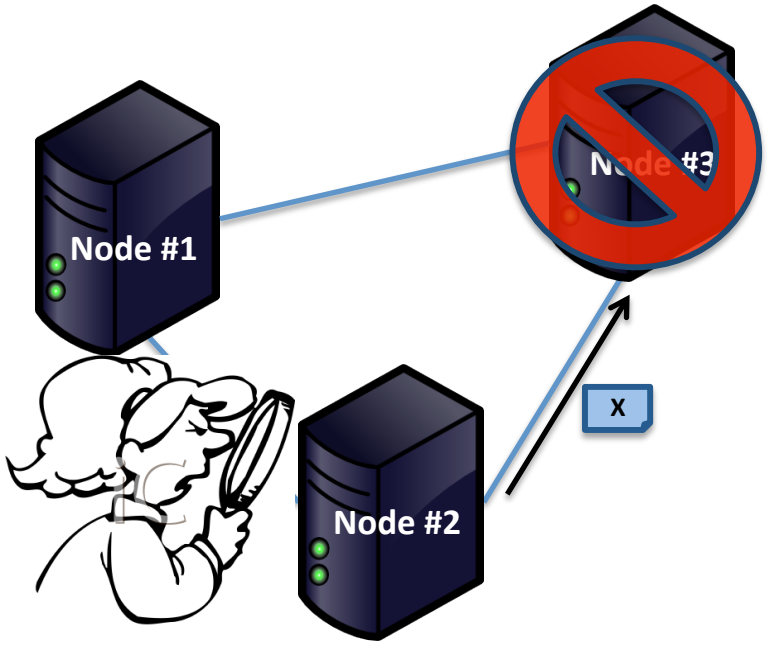
- Where is the program location that ‘x’ defined?
  -> From network..
- Which node sent this message?

Introduction



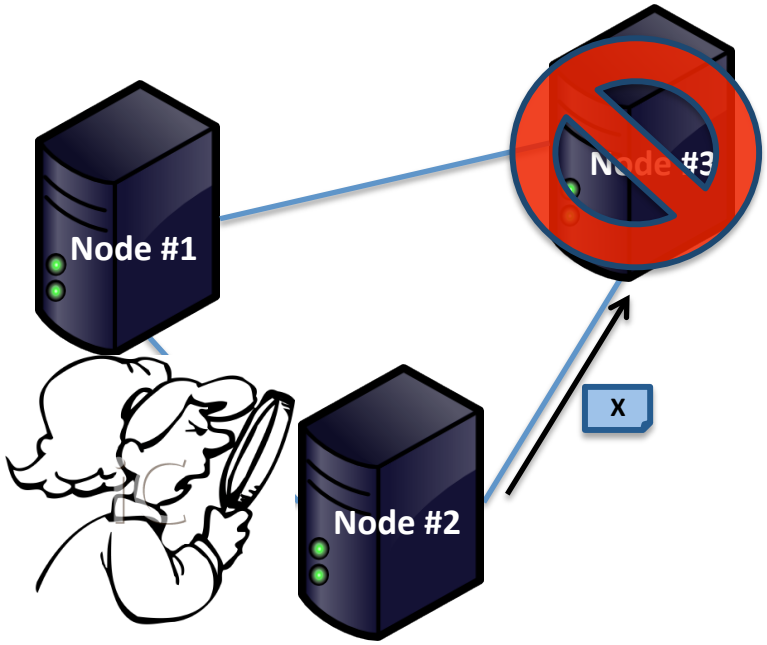
- Where is the program location that ‘x’ defined?
  -> From network..
- Which node sent this message?
  -> Node #2

Introduction



- Where is the send point?

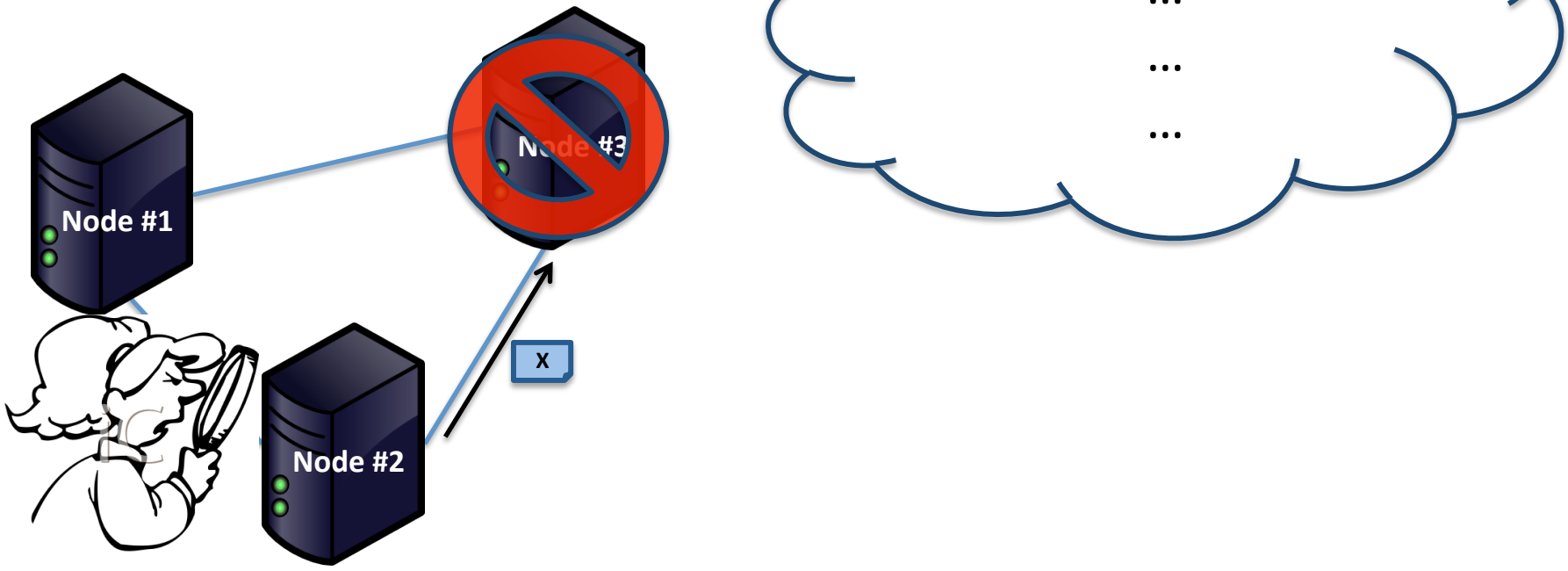
Introduction



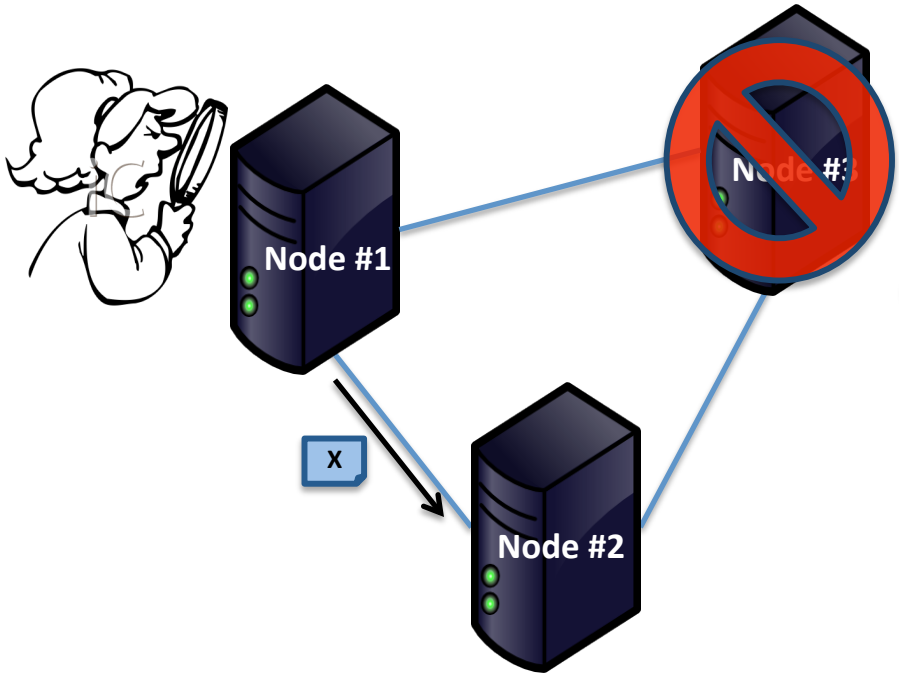
- Where is the send point?
  -> Which "send"?

```
Send(node#3, …);
        …
Send(node#3, …);
        …
Send(node#3, …);
        …
Send(node#3, …);
        …
Send(node#3, …);
        …
```

Introduction



Introduction



Not easy and time consuming!

Recon : Our approach

- Unified debugging service
  - Recording and replaying technique

Recon : Our approach

- Unified debugging service
  - Recording and replaying technique
  - Supports SQL-like queries for describing different levels of system artifacts
    - Nodes, Communication channels,
    - Event causality, instruction, etc

Recon : Our approach

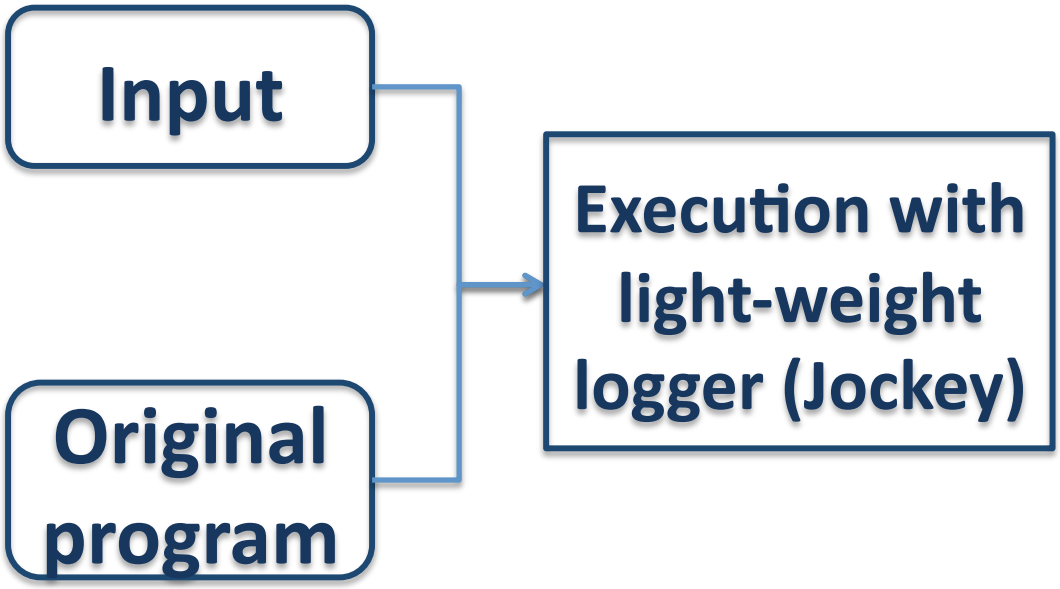
- Unified debugging service
  - Recording and replaying technique
  - Supports SQL-like queries for describing different levels of system artifacts
    - Nodes, Communication channels,
    - Event causality, instruction, etc
  - Dynamically instrument the program on demand to answer the query

Recon : Recording overview

Input

Original program

Recon : Recording overview



Recon : Recording overview

Recon : Replay overview

Original program

Queries

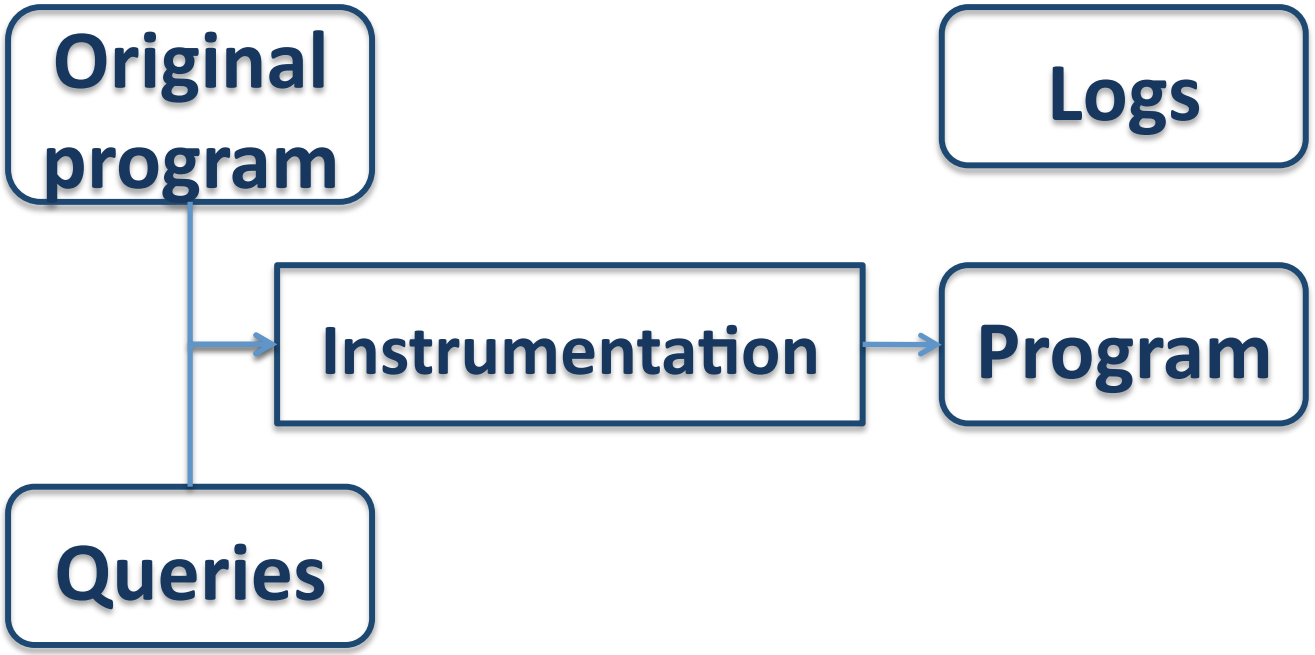
Recon : Replay overview

Original program

Queries

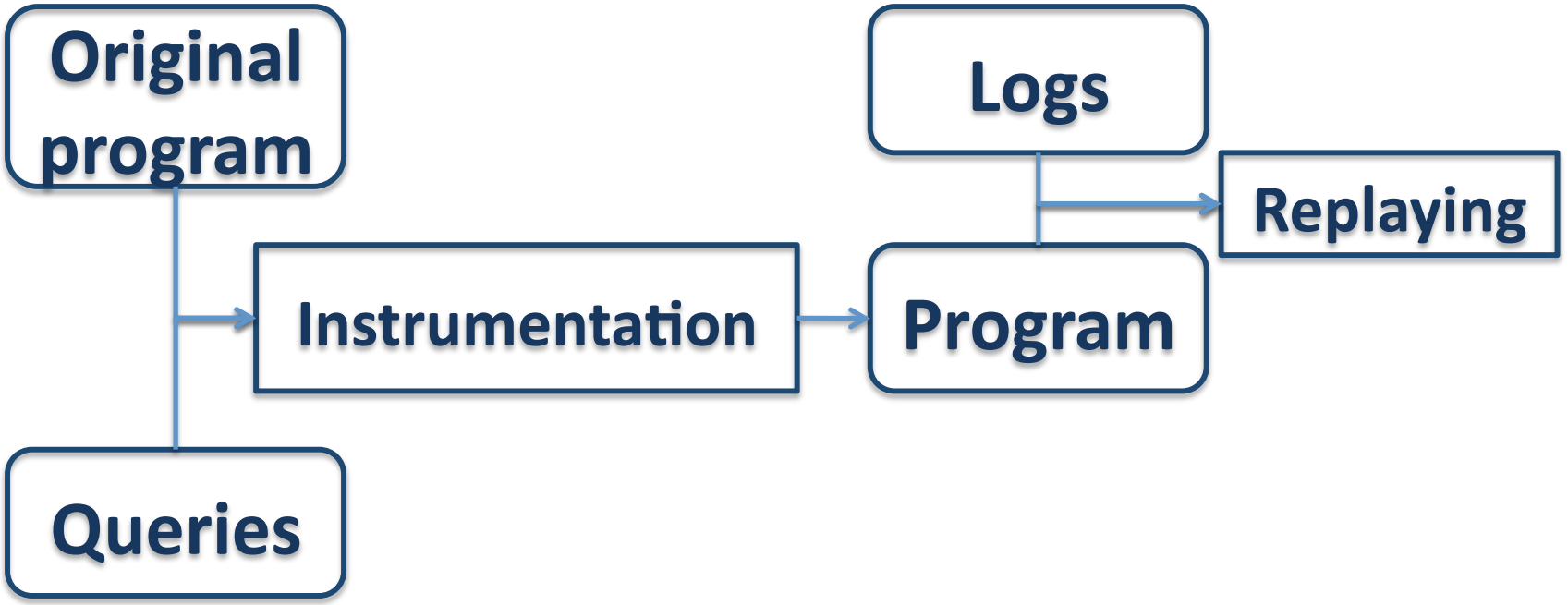
- Compile query to decide **what level of instrumentation** to collect the queried information

Recon : Replay overview



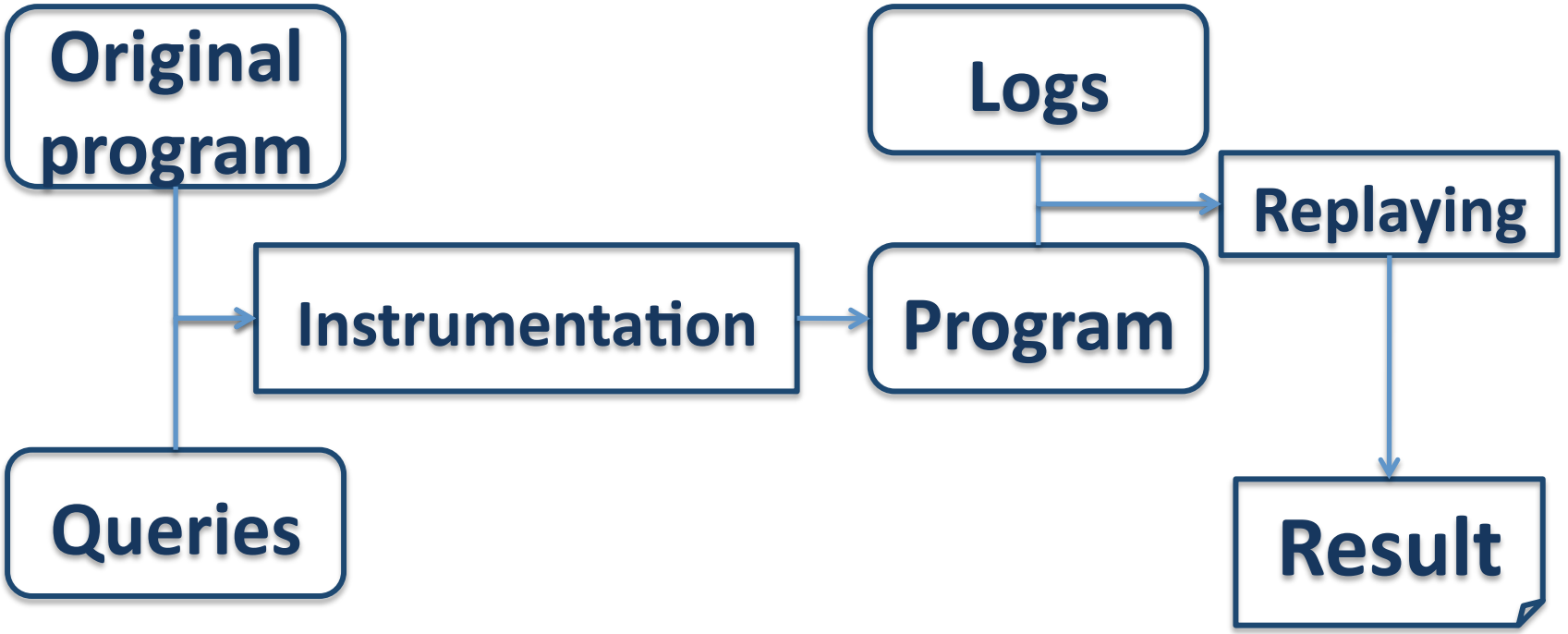
PIN : Dynamic instrumentation tool

Recon : Replay overview

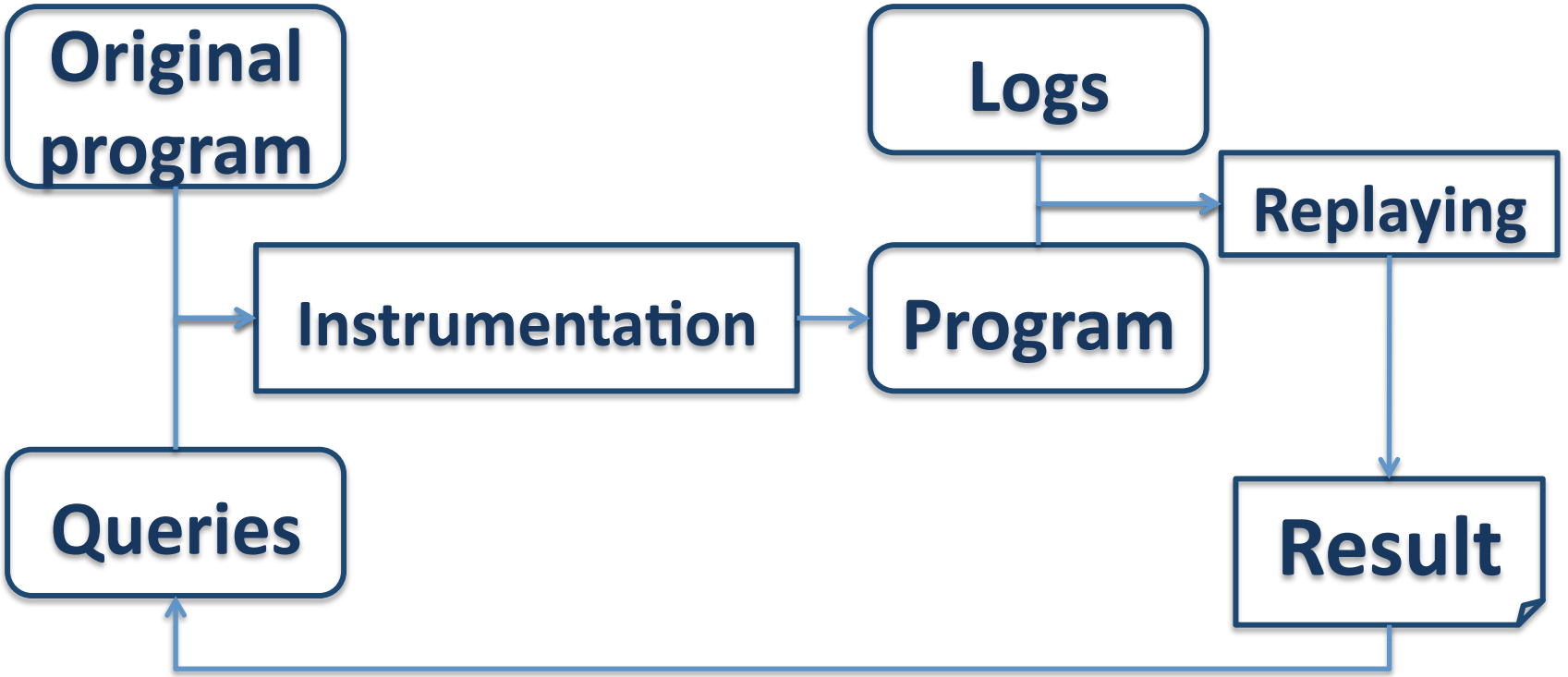


PIN : Dynamic instrumentation tool

Recon : Replay overview

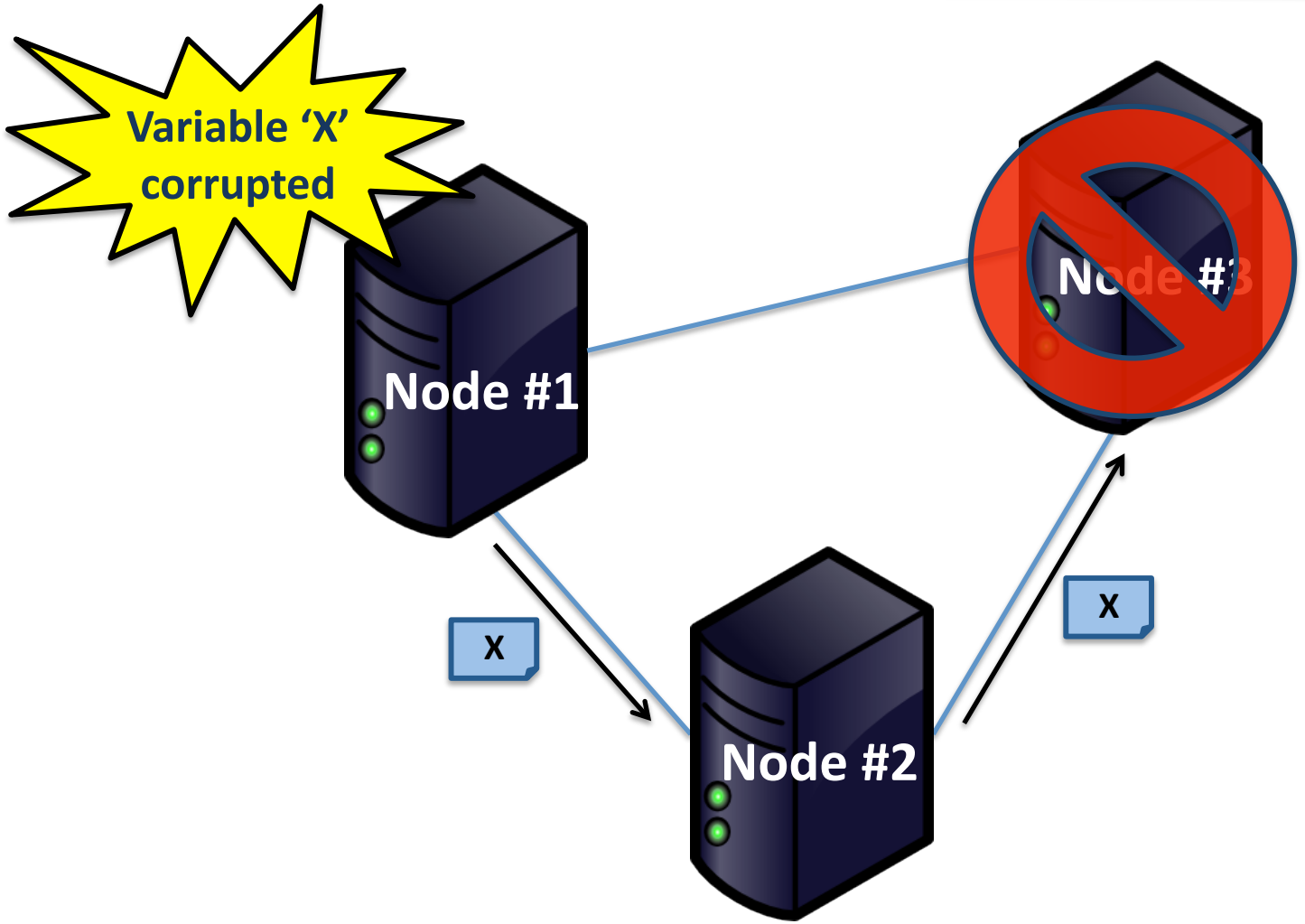


Recon : Replay overview

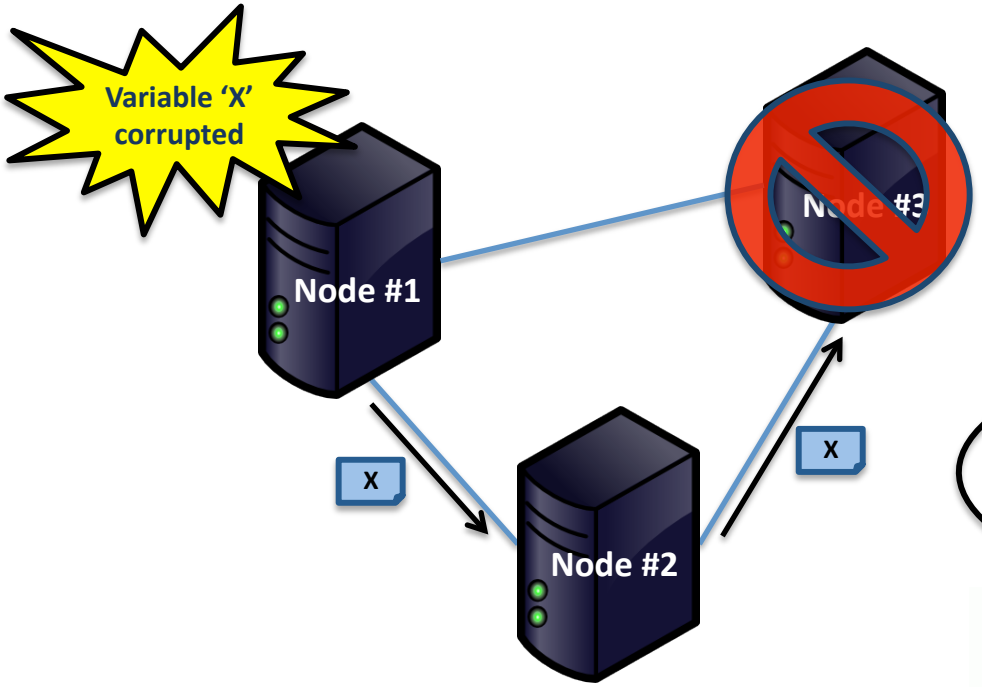


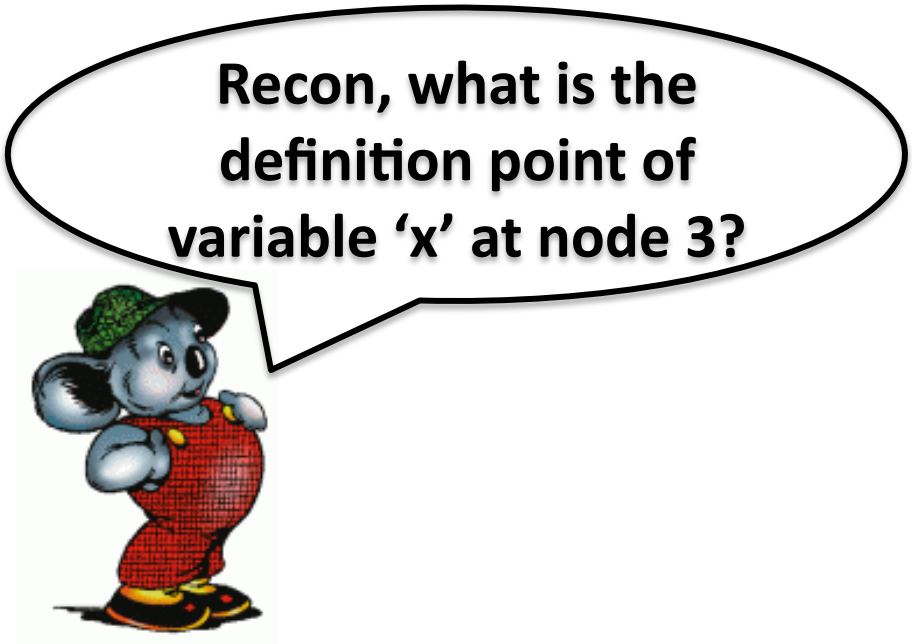
Refine query based on previous result

Recon : Our approach

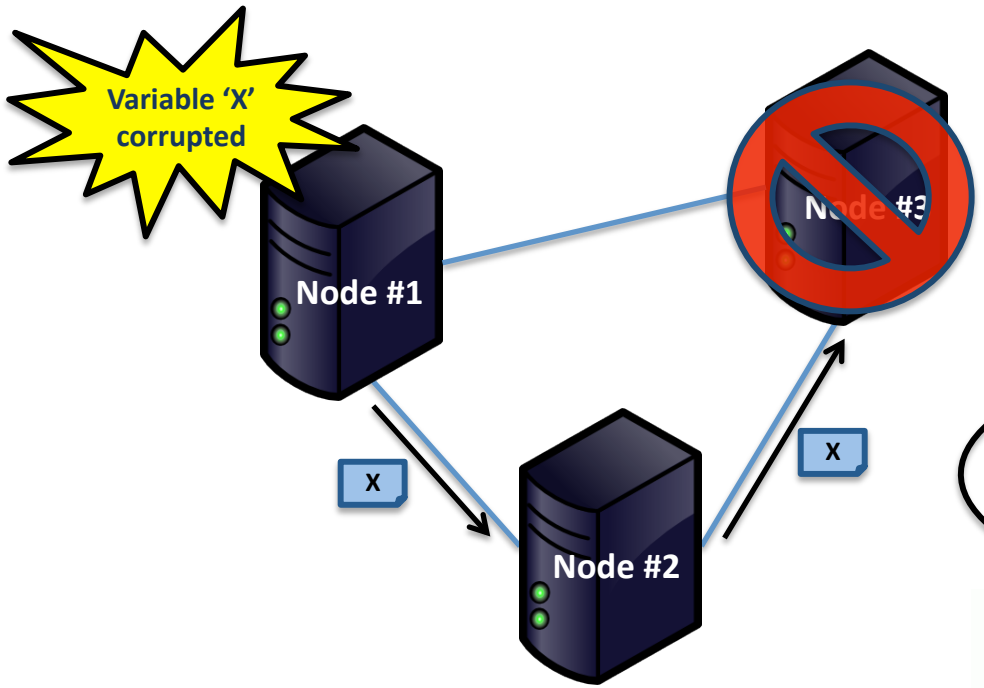


Recon : Our approach





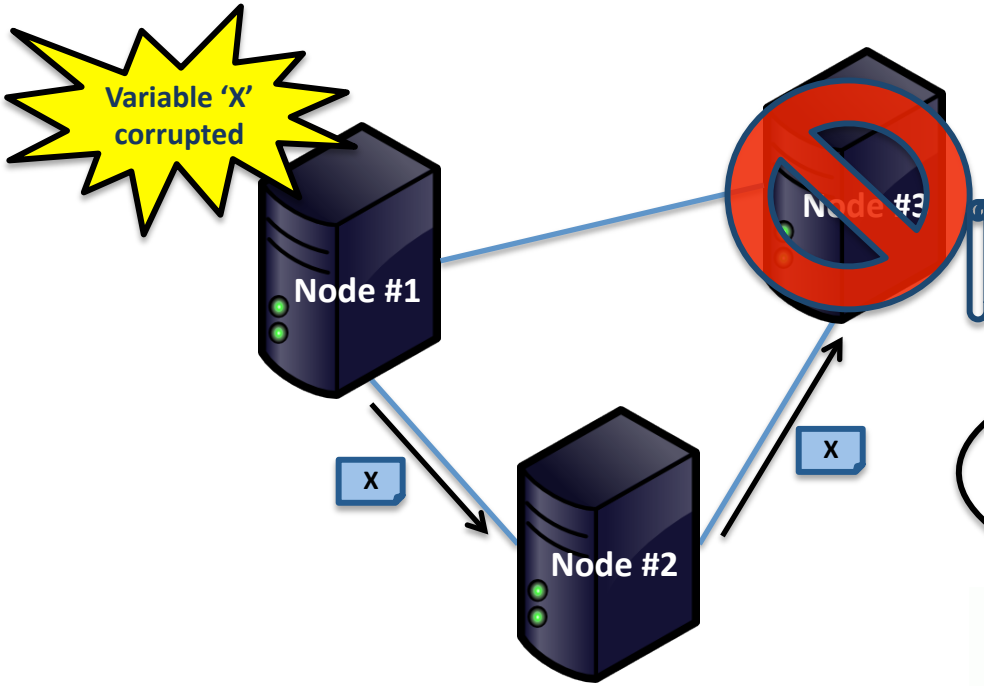
Recon : Our approach



SELECT **def_point** FROM **Data_Dependence** WHERE **variable = x** AND **use = "crash_point"** AND **host = 3**

Recon, what is the definition point of variable 'x' at node 3?

Recon : Our approach

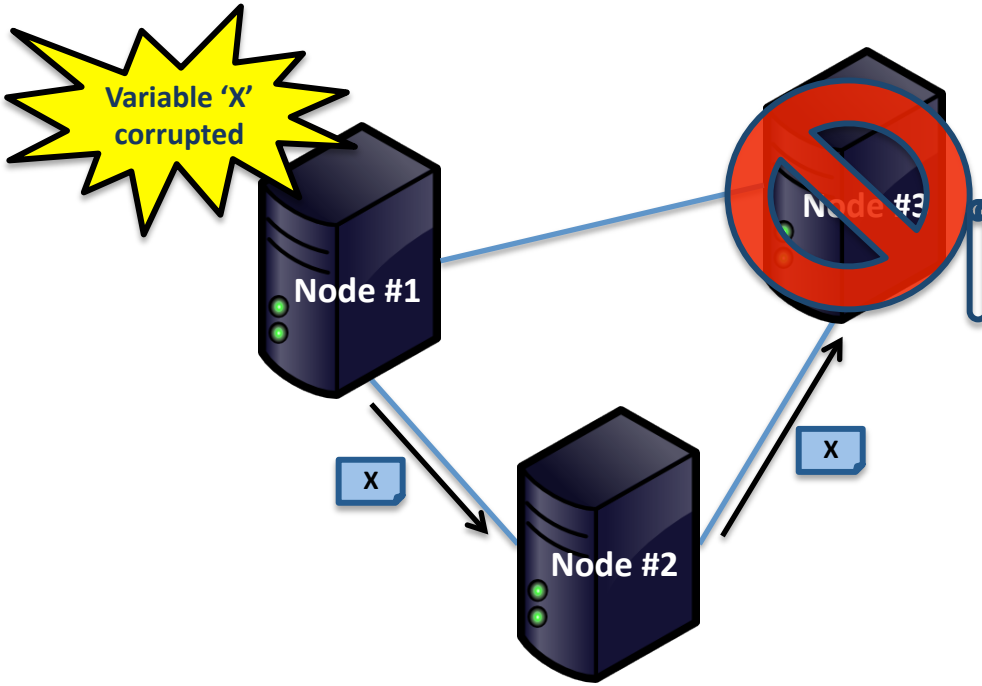


SELECT **def_point** FROM **Data_Dependence** WHERE **variable = x** AND **use =** **"crash_point"** AND **host = 3**

Recon : def_point=(d1.c:1028, 1)

Recon, what is the definition point of variable 'x' at node 3?

Recon : Our approach



SELECT **def_point** FROM **Data_Dependence** WHERE **variable = x** AND **use = "crash_point"** AND **host = 3**

Recon : def_point=(d1.c:1028, 1)

SELECT **sender, send_point** FROM **Communication** WHERE **receiver=3** and **recv_point = (d1.c:1028, 1)**

Recon : sender=2,
send_point = (d2.c:21, 1)

SQL Interface

- Predefined relations that describe different levels of distributed system aspects
  - Output
  - State
  - Communication
  - Control dependence
  - Data Dependence
  - Control flow

SQL Interface

- Predefined relations that describe different levels of distributed system aspects
  - Output
  - State
  - Communication

  } High-level relations

  - Control dependence
  - Data Dependence
  - Control flow

  } Low-level relations

SQL Interface

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

At the "**host**", output "**value**" is generated at the "**location**".

SQL Interface

(Source file, line number, instance)

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

SQL Interface

(Source file, line number, instance)

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SQL Interface

(Source file, line number, instance)

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **location** FROM **OUT**
WHERE **value = "ERROR"** AND
host = 0

SQL Interface

(Source file, line number, instance)

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **location** FROM **OUT**
WHERE **value = "ERROR"** AND
host = 0

Recon : (f.c, 12, 1)

SQL Interface


```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

Control dependence (CD)

Field	Type	Description
host	INT	Host ID
branch	EXE_PNT	Branch point
location	EXE_PNT	Execution point

At the "**host**", execution of the "**location**" is controlled by "**branch**".

SQL Interface

Control dependence (CD)

Field	Type	Description
host	INT	Host ID
branch	EXE_PNT	Branch point
location	EXE_PNT	Execution point

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **branch** FROM **CD**
WHERE **location = (f.c,12,1)**
AND **host = 0**

Recon : (f.c,11,1)

SQL Interface

Data dependence (DD)

Field	Type	Description
host	INT	Host ID
definition	EXE_PNT	Definition point
use	EXE_PNT	Use point
variable	STRING	Variable of dependence

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

At the "**host**", the "**variable**" at "**use**" point is defined by "**definition**" point.

SQL Interface

Data dependence (DD)

Field	Type	Description
host	INT	Host ID
definition	EXE_PNT	Definition point
use	EXE_PNT	Use point
variable	STRING	Variable of dependence

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **definition** FROM **DD**
WHERE **variable = x** AND
use = (f.c,11,1) AND **host = 0**

Recon : (f.c,10,1)

SQL Interface

Communication (COM)

Field	Type	Description
sender	INT	Sender ID
send_point	EXE_PNT	Send point
receiver	INT	Receiver ID
recv_point	EXE_PNT	Receive point
message	BYTE[]	message

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **sender, send_point**
FROM **COM** WHERE **receiver =0**
AND **recv_point = (f.c,10,1)**

SQL Interface

Communication (COM)

Field	Type	Description
sender	INT	Sender ID
send_point	EXE_PNT	Send point
receiver	INT	Receiver ID
recv_point	EXE_PNT	Receive point
message	BYTE[]	message

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **sender, send_point**
FROM **COM** WHERE **receiver =0**
AND **recv_point = (f.c,10,1)**

```
<f1.c>
5 send(fd, &y, ..);
```

Recon : sender = 1,
send_point = (f1.c,5,1)

State (ST)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Execution point
variable	STRING	Variable
value	BYTE[]	Untyped value

Control dependence (CD)

Field	Type	Description
host	INT	Host ID
branch	EXE_PNT	Branch point
location	EXE_PNT	Execution point

Control Flow (CF)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Execution point
pc	INT	Program counter
instance	INT	Instance
file	STRING	Source file
line	INT	Line number

Data dependence (DD)

Field	Type	Description
host	INT	Host ID
definition	EXE_PNT	Definition point
use	EXE_PNT	Use point
variable	STRING	Variable of dependence

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

Communication (COM)

Field	Type	Description
sender	INT	Sender ID
send_point	EXE_PNT	Send point
receiver	INT	Receiver ID
recv_point	EXE_PNT	Receive point
message	BYTE[]	message

SQL Interface

Output (OUT)

Field	Type	Description
host	INT	Host ID
location	EXE_PNT	Output point
value	BYTE[]	Output value

Control dependence (CD)

Field	Type	Description
host	INT	Host ID
branch	EXE_PNT	Branch point
location	EXE_PNT	Execution point

```
<f.c>
10 recv(fd, &x,); /* x=11 */
11 if(x > 10)
12   printf("ERROR");
```

SELECT **CD.branch** FROM **CD** AND **OUT** WHERE **CD.location = OUT.location** AND **OUT.value = "ERROR"** AND **host = 0**

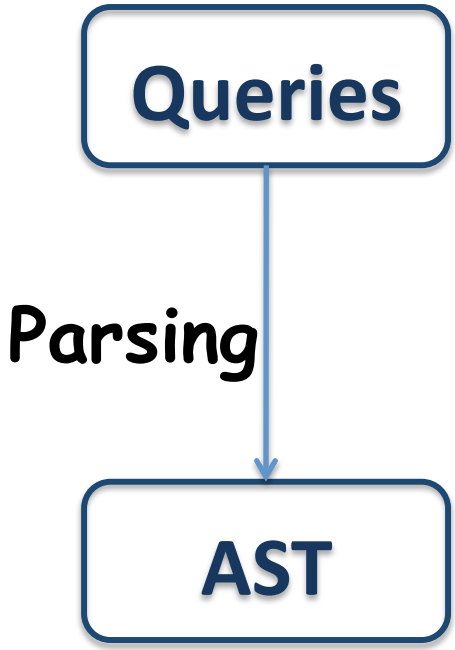
Recon : (f.c,11,1)

Query compiler

- Compile query to instrumentation
- Replay with the instrumentation answers the query

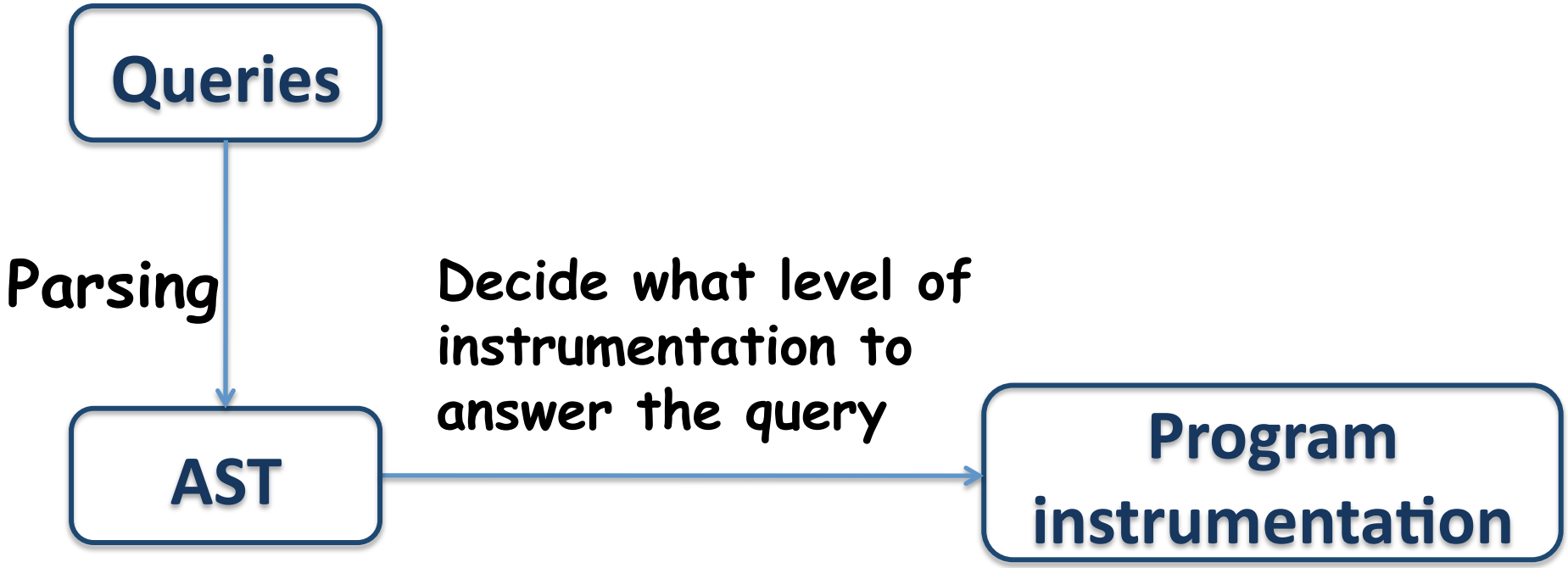
Query compiler

- Compile query to instrumentation
- Replay with the instrumentation answers the query



Query compiler

- Compile query to instrumentation
- Replay with the instrumentation answers the query



Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

Query compiler

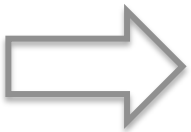
- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



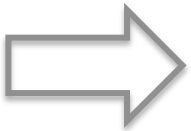
Instrumented code

```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

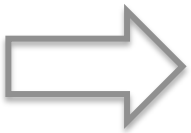
```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

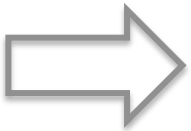
```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2   s1;
3   s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]
(cd.c,1,1)

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

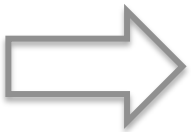
```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2   s1;
3   s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]
(cd.c,1,1)

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

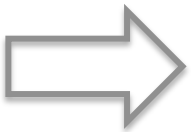
```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]
(cd.c,1,1)

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

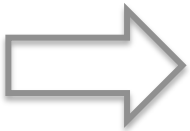
[CD_stack]
(cd.c,1,1)

Dependence on (cd.c,3,1)?
Stack top = (cd.c,1,1)

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

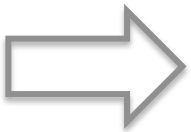
```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

```
<cd.c>
1  if(p) {
2    s1;
3    s2;
4  }
5  s3;
```



Instrumented code

```
<cd.c>
1 if(p) {
 CD_stack.push(cd.c,1,1);
2  s1;
3  s2;
 CD_stack.pop();
4 }
5 s3;
```

[CD_stack]

Dependence on (cd.c,5,1)?

Stack top = *none*

Query compiler

- Instrumentation to answer a query on the CD
  - Detect branch instruction and post-dominator

Recon does NOT collect trace.
Answer the query on the fly.

```
4 }
5 s3;
```

Dependence on (cd.c,5,1)?
Stack top = *none*

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

Query compiler

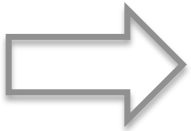
- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



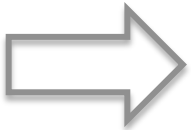
Instrumented code

```
<dd.c>
1 x = 10;
  shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
  shadow_mem(y) = (dd.c,2,1);
```

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



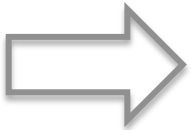
Instrumented code

```
<dd.c>
1 x = 10;
  shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
  shadow_mem(y) = (dd.c,2,1);
```

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



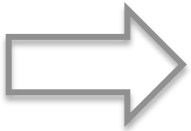
Instrumented code

```
<dd.c>
1 x = 10;
   shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
   shadow_mem(y) = (dd.c,2,1);
```

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



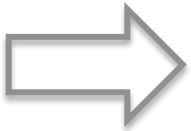
Instrumented code

```
<dd.c>
1 x = 10;
  shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
  shadow_mem(y) = (dd.c,2,1);
```

Query compiler

- Instrumentation to answer a query on the DD
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



Instrumented code

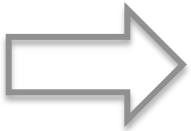
```
<dd.c>
1 x = 10;
  shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
  shadow_mem(y) = (dd.c,2,1);
```

Dependence on x?
= shadow_mem(x) = (dd.c,1,1)

Query compiler

- Instrumentation to answer a DD query
  - Detect dependence through "shadow memory"

```
<dd.c>
1   x = 10;
2   y = x+1;
```



Instrumented code

```
<dd.c>
1 x = 10;
  shadow_mem(x) = (dd.c,1,1);
2 y = x+1;
  shadow_mem(y) = (dd.c,2,1);
```

Dependence on x?
= shadow_mem(x) = (dd.c,1,1)

  - We developed an instrumentation module for PIN

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log

[System call Log]

…
sock = socket(..)
…
connect(sock, address, …)
…
recv(sock, …)

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log

[System call Log]

…
sock = socket(..)
…
connect(sock, address, …)
…
recv(sock, …)

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log

[System call Log]

…
sock = socket(..)
…
connect(sock, address, …)
…
recv(sock, …)

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point

[System call Log] – Sender

…
connect(sock, address, …)
…
ret = send(sock, …)
…
ret = send(sock, …)

[System call Log] - Receiver

…
connect(sock, address, …)
…
ret = recv(sock, msg1, …)
…
ret = recv(sock, msg2, …)

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point

[System call Log] – Sender

…
connect(sock, address, …)
…
ret = send(sock, …)
…
ret = send(sock, …)

[System call Log] - Receiver

…
connect(sock, address, …)
…
ret = recv(sock, msg1, …)
…
ret = recv(sock, msg2, …)

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point

[System call Log] – Sender

…
connect(sock, address, …)
…
ret = send(sock, …)
…
ret = send(sock, …)

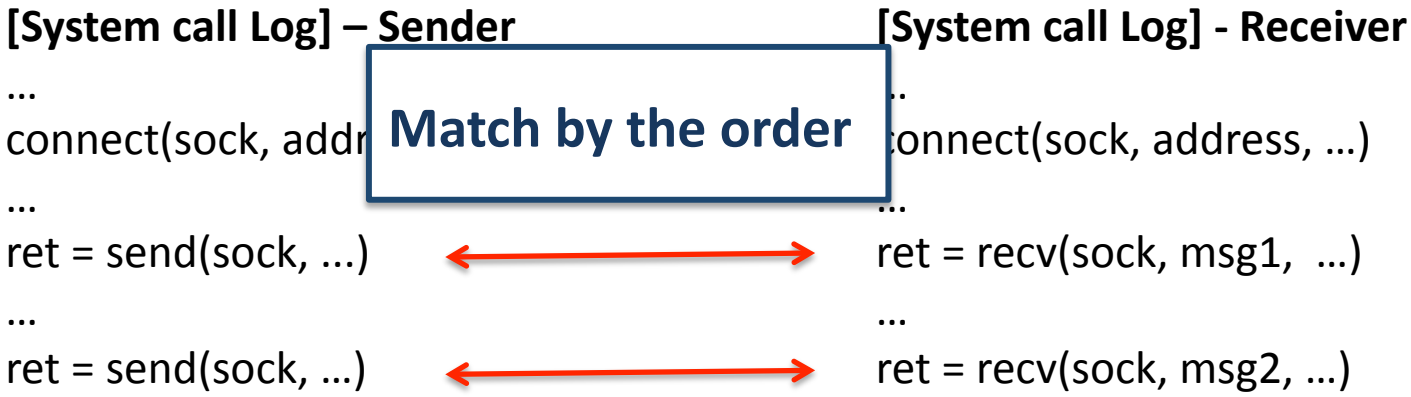
[System call Log] - Receiver

…
connect(sock, address, …)

We do not log the message for "send" system call
- Not necessary to replay deterministically
- Minimize the overhead

Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point



Query compiler

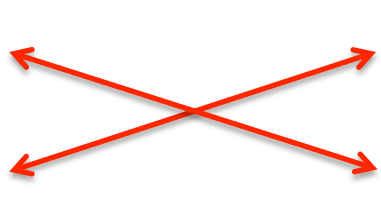
- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point

[System call Log] – Sender

…
connect(sock, address, …)
…
ret = send(sock, **msg1**, ..)
…
ret = send(sock, **msg2**, ..)

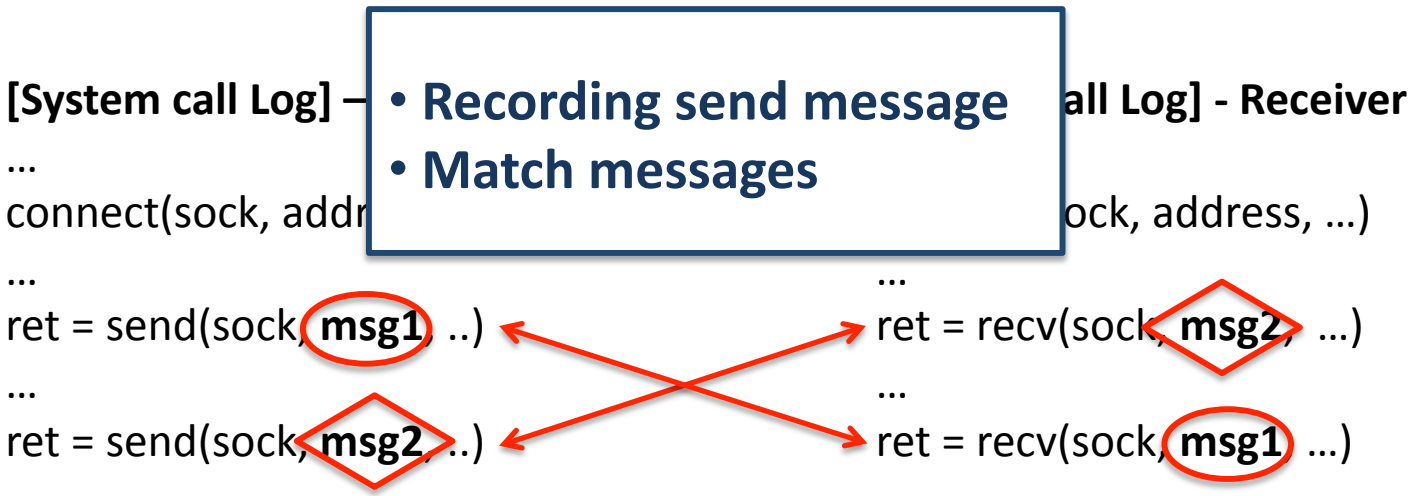
[System call Log] - Receiver

…
connect(sock, address, …)
…
ret = recv(sock, **msg2**, …)
…
ret = recv(sock, **msg1**, …)



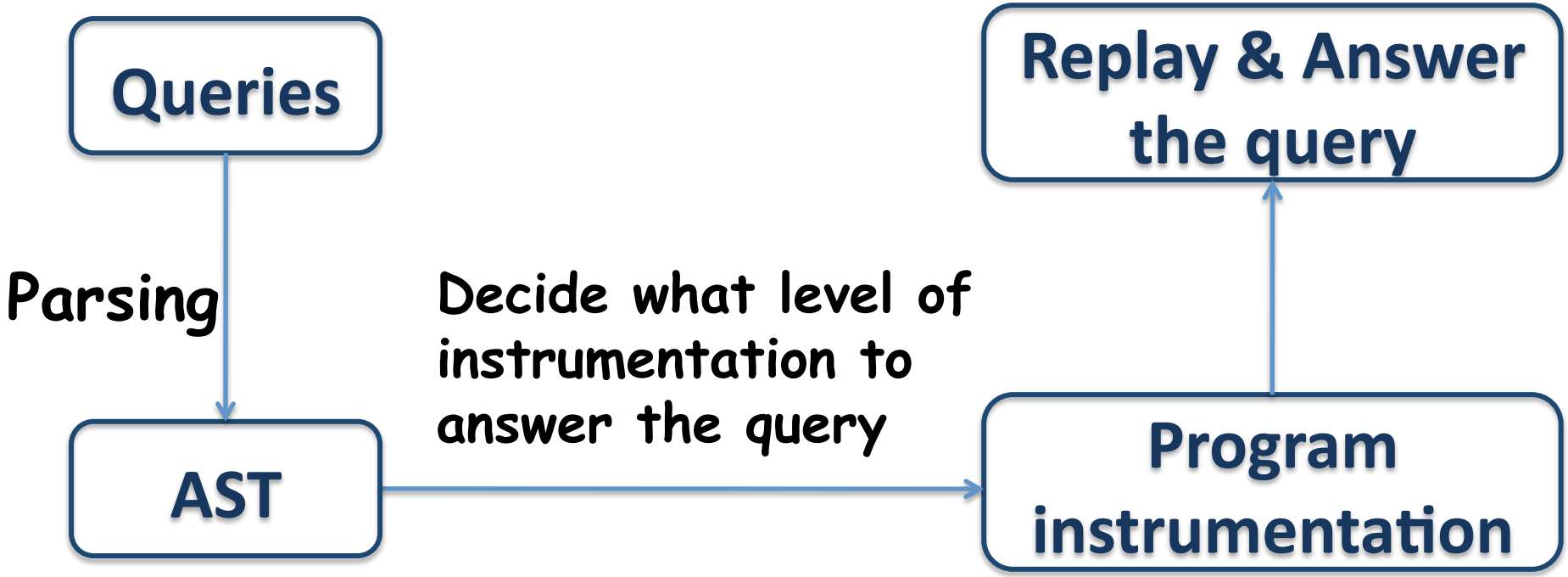
Query compiler

- To answer a query on the COM relation
  - Finding sender node
    - Analyze the system call log
  - Finding send_point
    - Match the send point and the received point



Query compiler

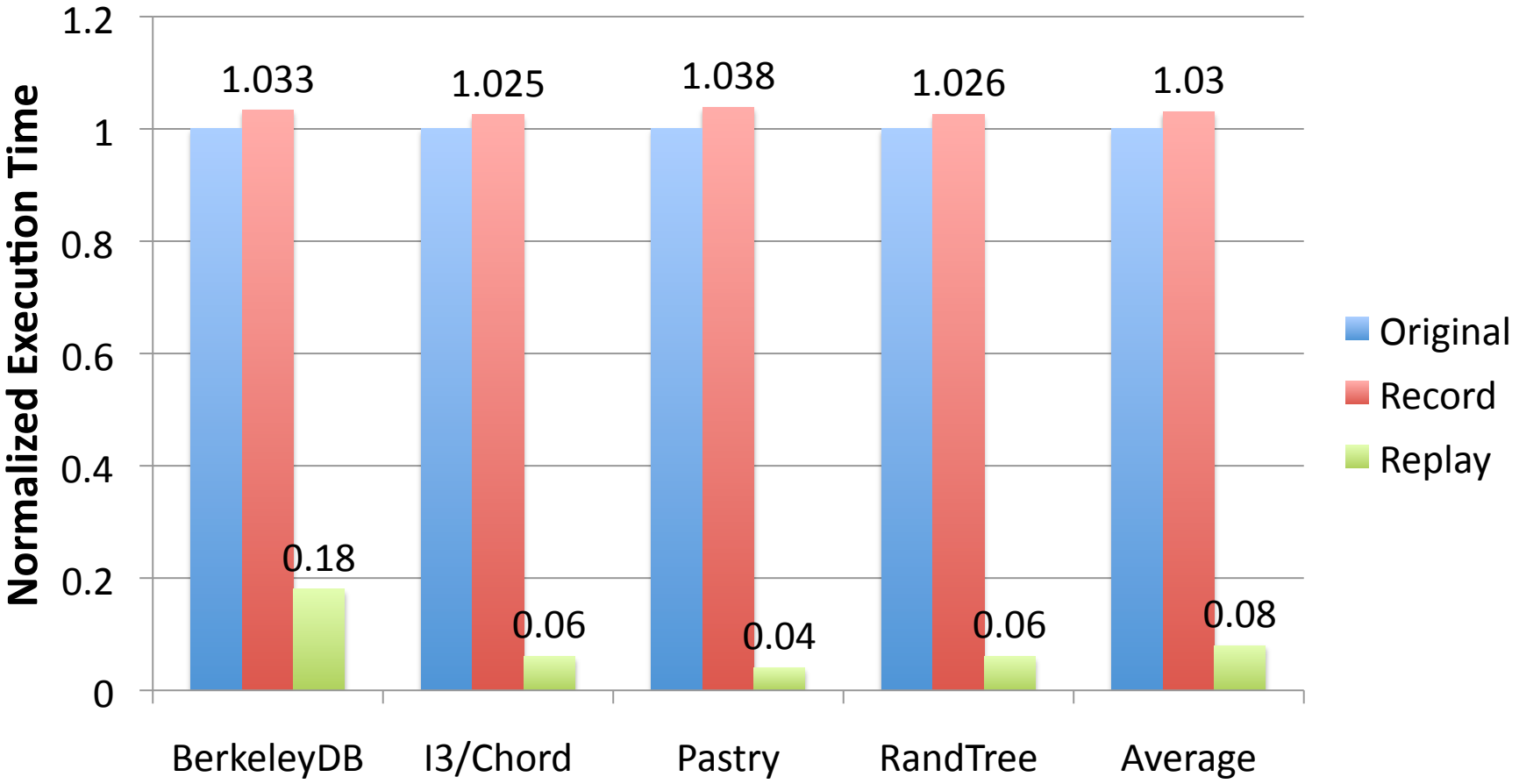
- Compile query to instrumentation
- Replay with the instrumentation answers the query



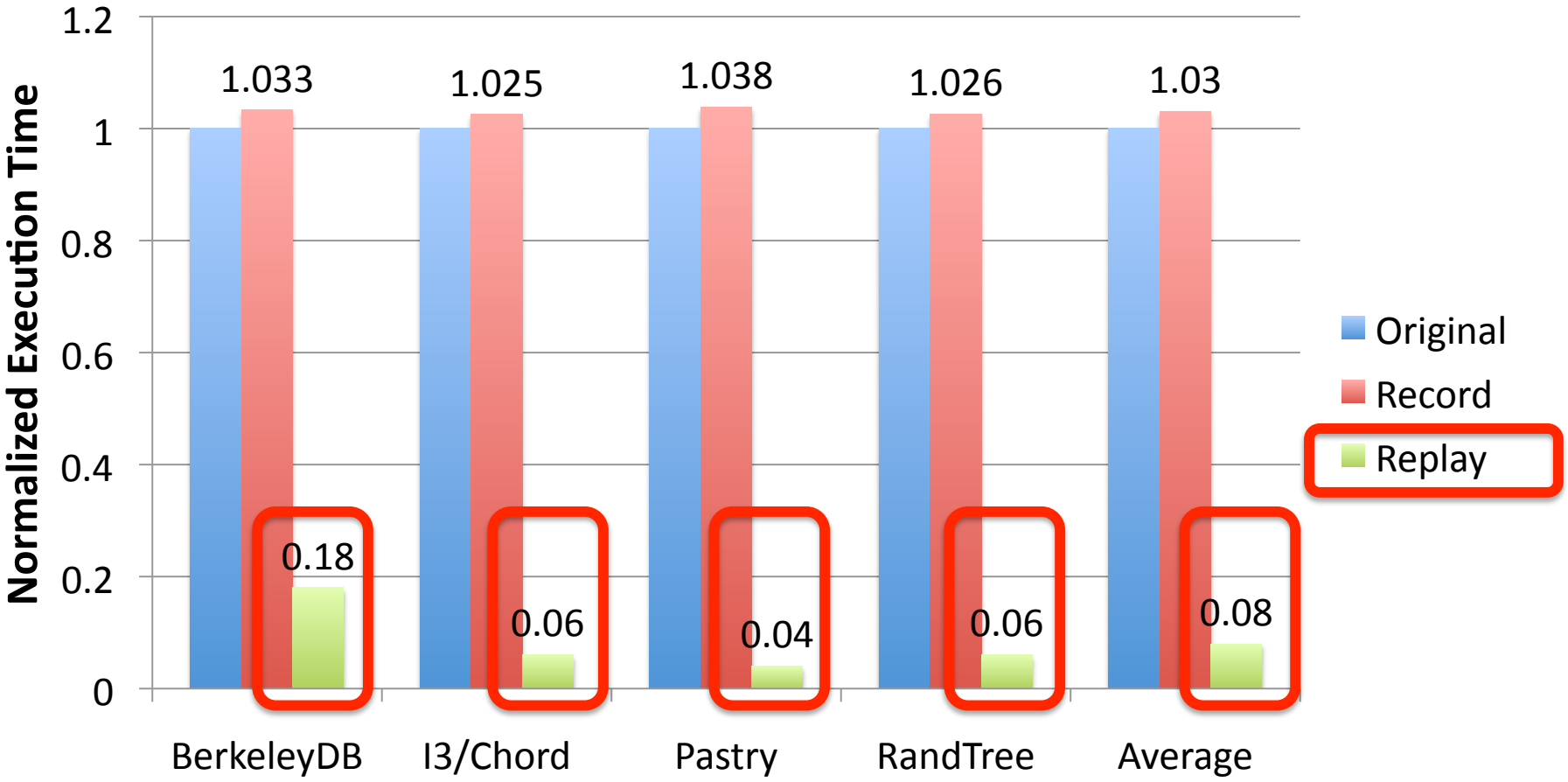
Evaluation

- Evaluate Recon with 8 real bugs from
  - BerkeleyDB
  - I3/Chord
  - Pastry
  - RandTree
- Overhead of Recon
  - Logging overhead
  - Replay overhead with different levels of instrumentation

Recording and replaying overhead



Recording and replaying overhead



Time saved by emulating all systemcalls

BerkeleyDB Logging Cost

# of nodes	Execution time (sec)		Logging Overhead		Log size
	Local	EC2	Local	EC2	
1	180.5	25.58	7.19%	2.88%	6.79MB
2	414.4	396.33	3.14%	0.54%	9.13MB
4	512.0	212.74	3.19%	1.11%	12.28MB
8	594.7	313.4	3.04%	1.05%	17.07MB

BerkeleyDB Logging Cost

# of nodes	Execution time (sec)		Logging Overhead		Log size
	Local	EC2	Local	EC2	
1	180.5	25.58	7.19%	2.88%	6.79MB
2	414.4	396.33	3.14%	0.54%	9.13MB
4	512.0	212.74	3.19%	1.11%	12.28MB
8	594.7	313.4	3.04%	1.05%	17.07MB

Local : Independent virtual machines in a system
EC2 : Amazon EC2 Cloud system

BerkeleyDB Logging Cost

# of nodes	Execution time (sec)		Logging Overhead		Log size
	Local	EC2	Local	EC2	
1	180.5	25.58	7.19%	2.88%	6.79MB
2	414.4	396.33	3.14%	0.54%	9.13MB
4	512.0	212.74	3.19%	1.11%	12.28MB
8	594.7	313.4	3.04%	1.05%	17.07MB

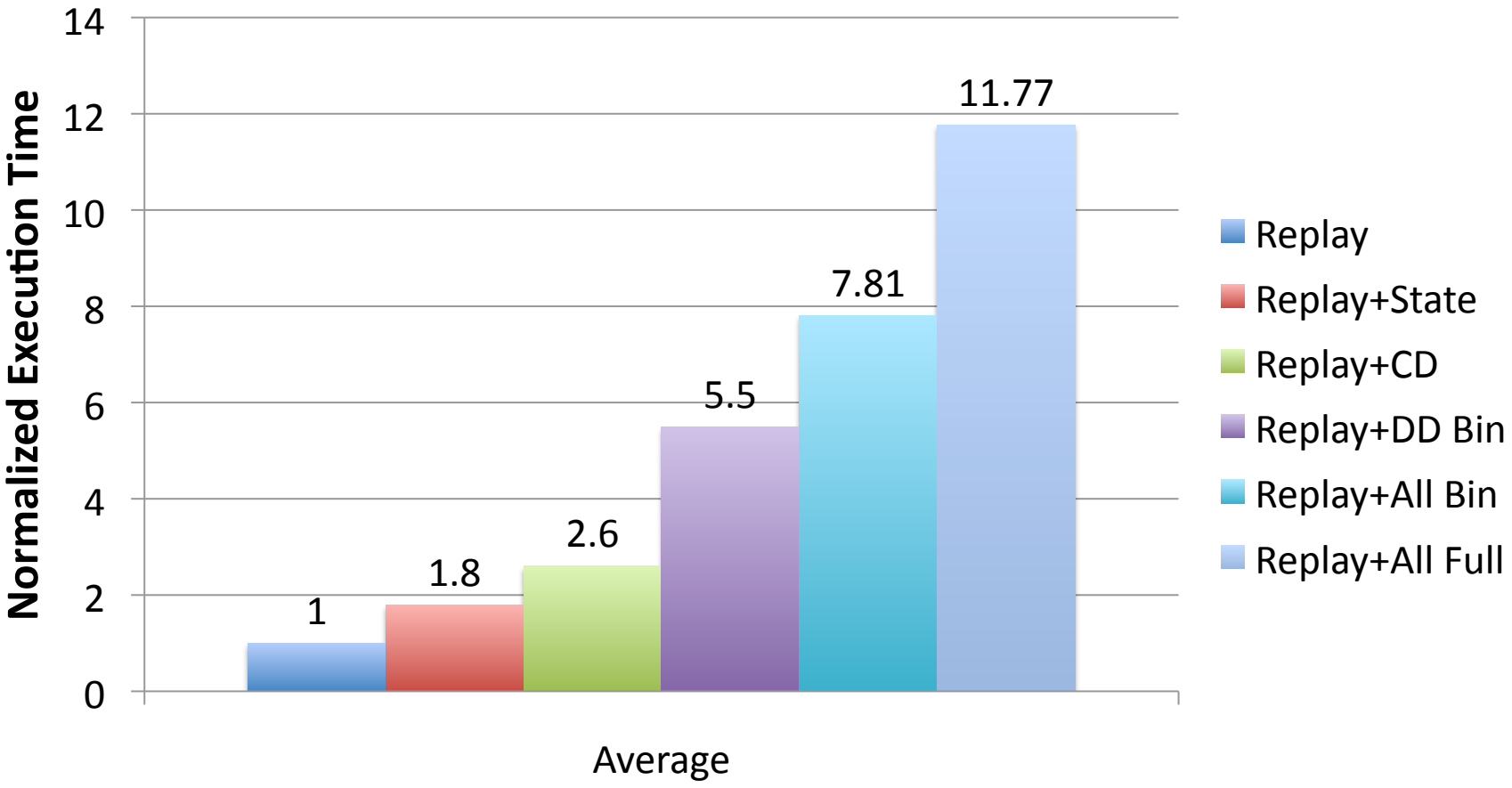
BerkeleyDB Logging Cost

# of nodes	Execution time (sec)		Logging Overhead		Log size
	Local	EC2	Local	EC2	
1	180.5	25.58	7.19%	2.88%	6.79MB
2	414.4	396.33	3.14%	0.54%	9.13MB
4	512.0	212.74	3.19%	1.11%	12.28MB
8	594.7	313.4	3.04%	1.05%	17.07MB

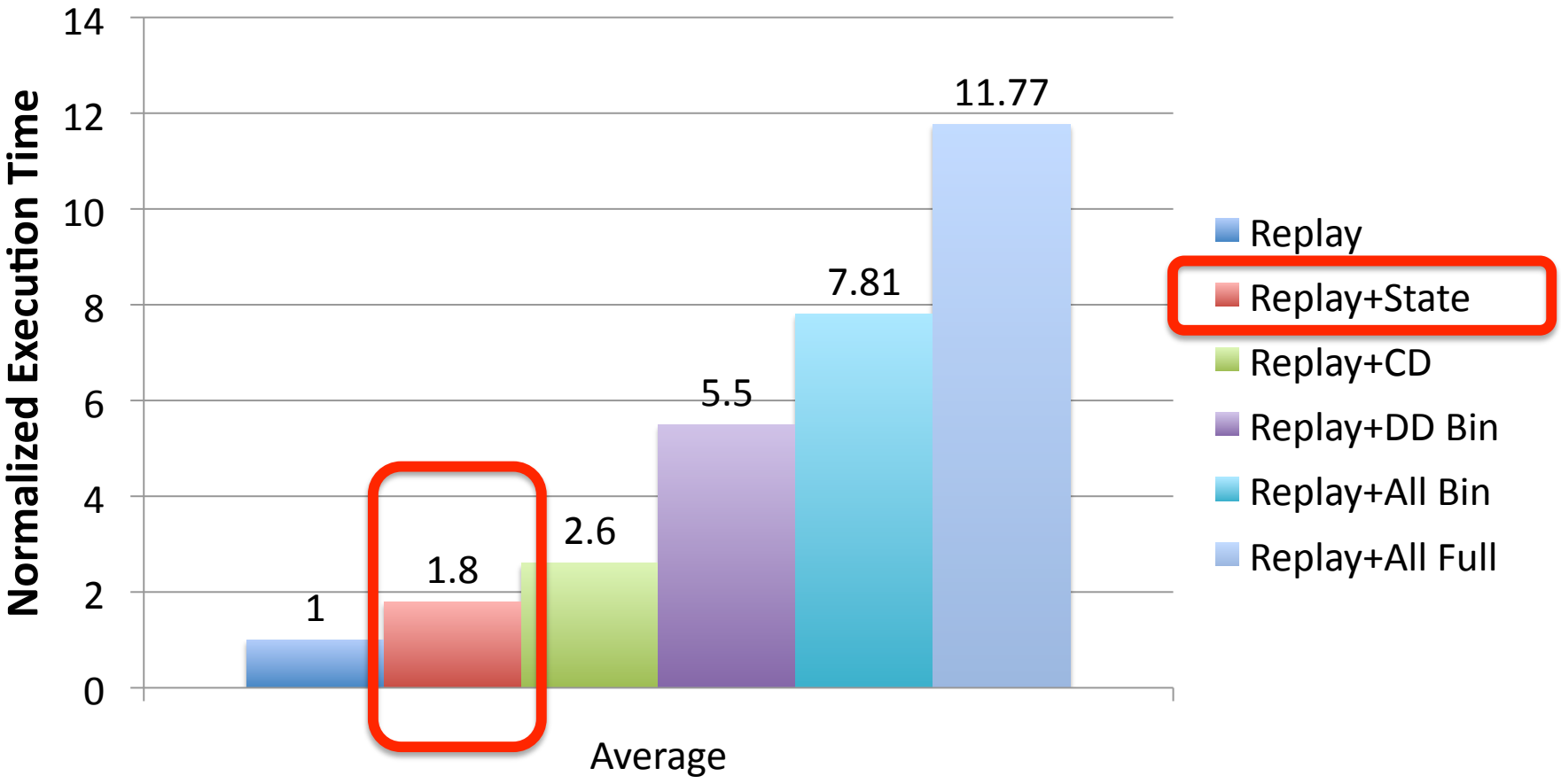
BerkeleyDB Logging Cost

# of nodes	Execution time (sec)		Logging Overhead		Log size
	Local	EC2	Local	EC2	
1	180.5	25.58	7.19%	2.88%	6.79MB
2	414.4	396.33	3.14%	0.54%	9.13MB
4	512.0	212.74	3.19%	1.11%	12.28MB
8	594.7	313.4	3.04%	1.05%	17.07MB

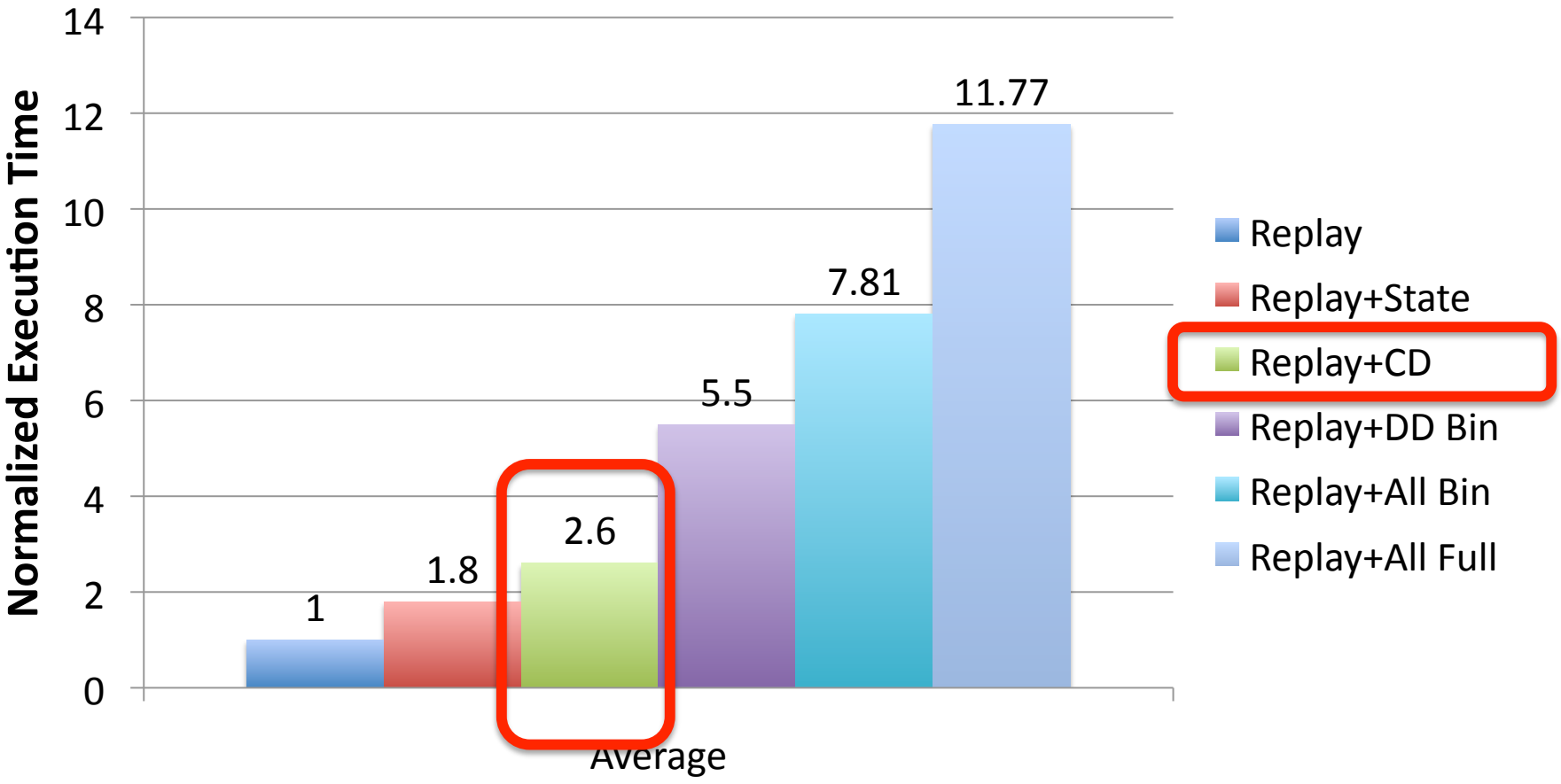
Instrumentation Overhead



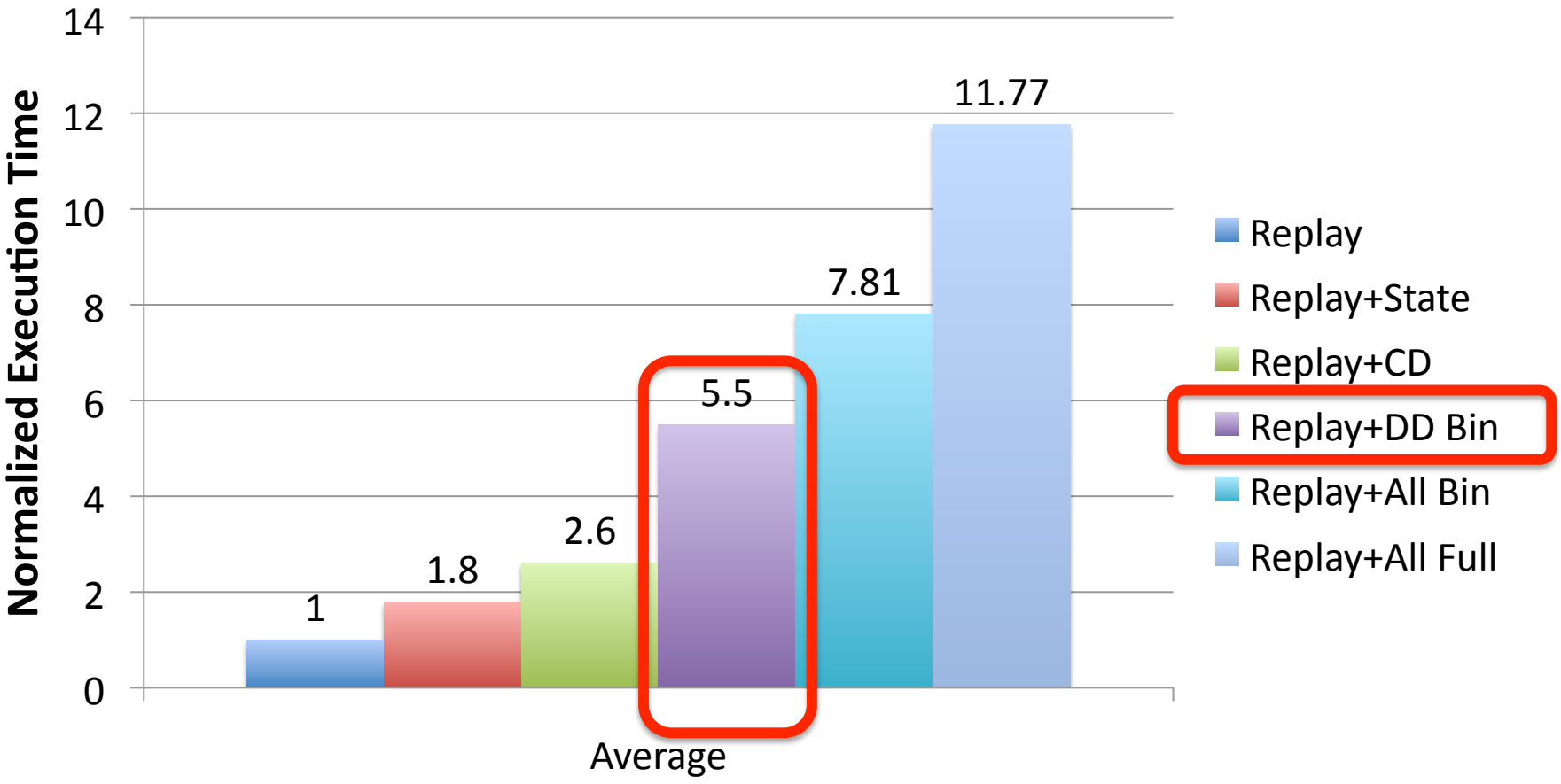
Instrumentation Overhead



Instrumentation Overhead



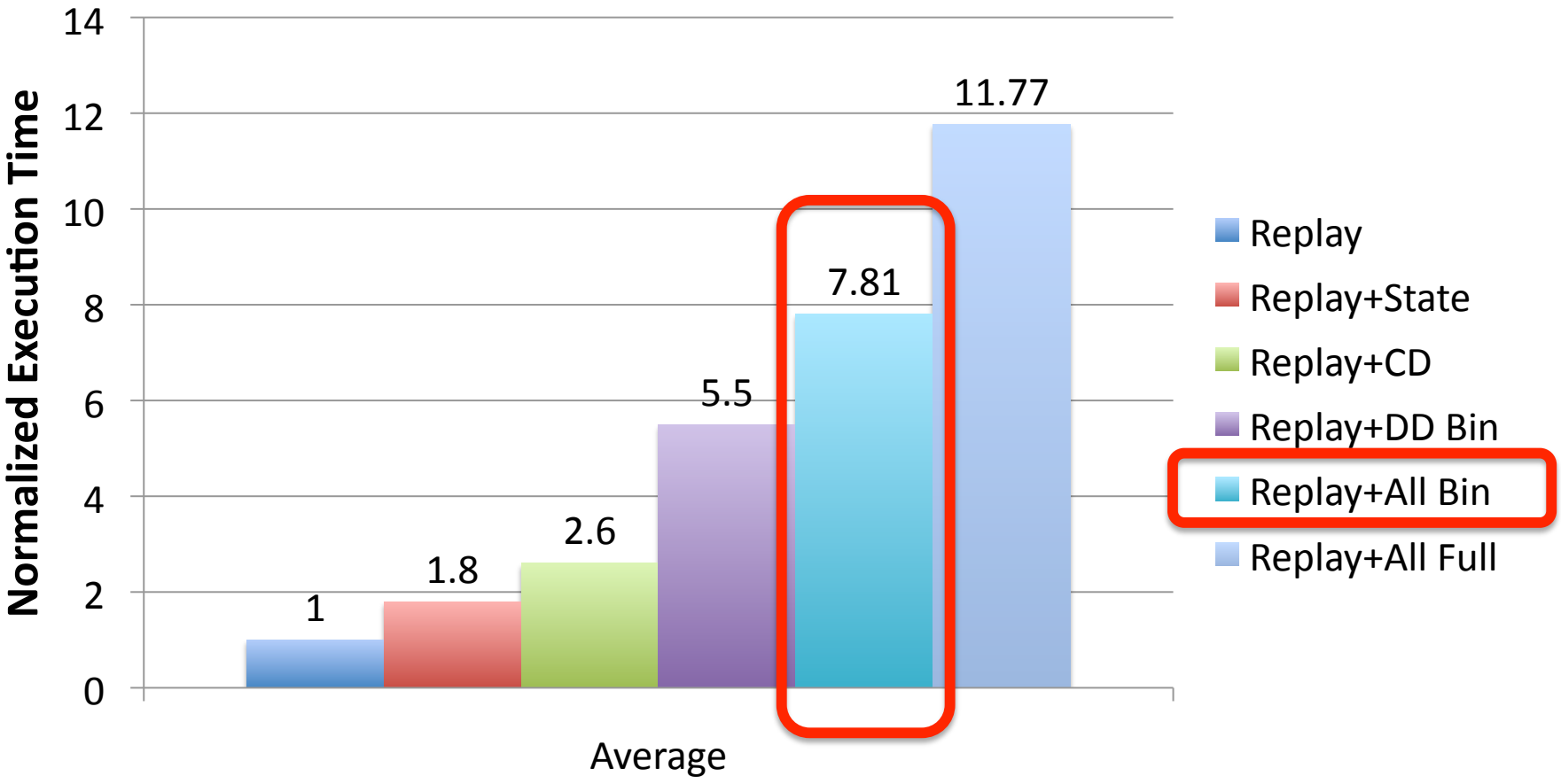
Instrumentation Overhead



Bin : Cover application binary

Full : Cover application binary and libraries

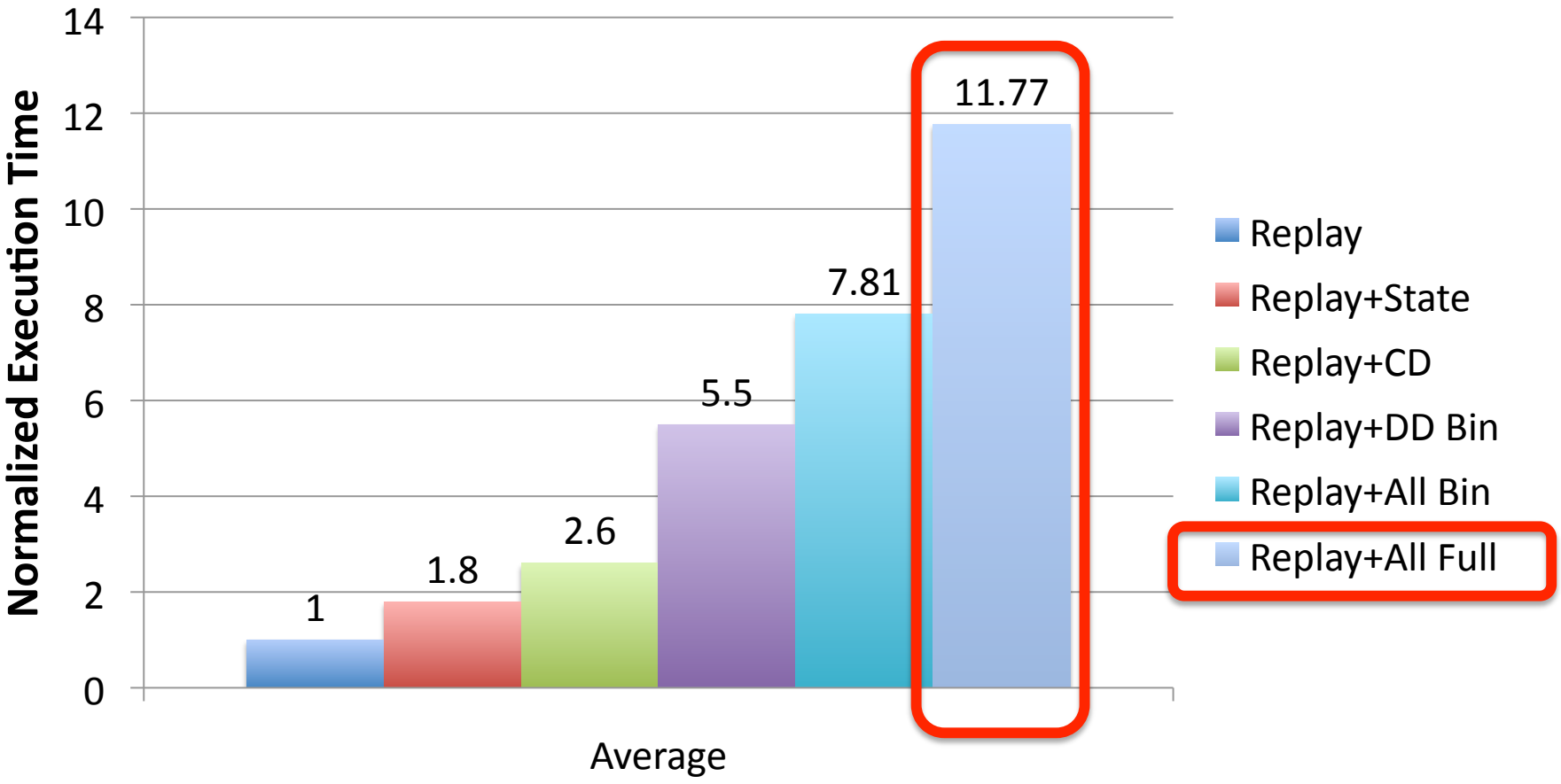
Instrumentation Overhead



Bin : Cover application binary

Full : Cover application binary and libraries

Instrumentation Overhead



Bin : Cover application binary

Full : Cover application binary and libraries

Analyzed bugs with queries

Bugs	Total Queries	STATE	CD	DD	OUTPUT	CF	COM	# of nodes
DB #1	13	1	4	6	1	0	2	2
DB #2	14	1	3	8	1	0	1	2
Chord #1	7	3	2	3	1	0	0	1
Chord #2	7	0	2	1	1	2	2	2
RandTree #1	9	0	2	4	1	0	3	3
RandTree #2	7	0	3	2	1	1	1	2
RandTree #3	5	1	1	3	1	0	1	2
Pastry	5	1	2	1	1	0	1	2

Analyzed bugs with queries

Bugs	Total Queries	STATE	CD	DD	OUTPUT	CF	COM	# of nodes
DB #1	13	1	4	6	1	0	2	2
DB #2	14	1	3	8	1	0	1	2
Chord #1	7	3	2	3	1	0	0	1
Chord #2	7	0	2	1	1	2	2	2
RandTree #1	9	0	2	4	1	0	3	3
RandTree #2	7	0	3	2	1	1	1	2
RandTree #3	5	1	1	3	1	0	1	2
Pastry	5	1	2	1	1	0	1	2

Analyzed bugs with queries

Bugs	Total Queries	STATE	CD	DD	OUTPUT	CF	COM	# of nodes
DB #1	13	1	4	6	1	0	2	2
DB #2	14	1	3	8	1	0	1	2
Chord #1	7	3	2	3	1	0	0	1
Chord #2	7	0	2	1	1	2	2	2
RandTree #1	9	0	2	4	1	0	3	3
RandTree #2	7	0	3	2	1	1	1	2
RandTree #3	5	1	1	3	1	0	1	2
Pastry	5	1	2	1	1	0	1	2

Analyzed bugs with queries

Bugs	Total Queries	STATE	CD	DD	OUTPUT	CF	COM	# of nodes
DB #1	13	1	4	6	1	0	2	2
DB #2	14	1	3	8	1	0	1	2
Chord #1	7	3	2	3	1	0	0	1
Chord #2	7	0	2	1	1	2	2	2
RandTree #1	9	0	2	4	1	0	3	3
RandTree #2	7	0	3	2	1	1	1	2
RandTree #3	5	1	1	3	1	0	1	2
Pastry	5	1	2	1	1	0	1	2

Analyzed bugs with queries

Bugs	Total Queries	STATE	CD	DD	OUTPUT	CF	COM	# of nodes
DB #1	13	1	4	6	1	0	2	2
DB #2	14	1	3	8	1	0	1	2
Chord #1	7	3	2	3	1	0	0	1
Chord #2	7	0	2	1	1	2	2	2
RandTree #1	9	0	2	4	1	0	3	3
RandTree #2	7	0	3	2	1	1	1	2
RandTree #3	5	1	1	3	1	0	1	2
Pastry	5	1	2	1	1	0	1	2

DB #1 bug : Permanent election failure
DB #2 bug : Master node panic

Case study

- BerkeleyDB Bug : Permanent election failure

<rep_elect.c>
368 if (send_vote == DB_EID_INVALID) {
371 __db_errx(env,
372 "No electable …",
<db_errx.c>
417 __db_errx(env, fmt, va_alist) {
435 sprintf(new_fmt, "%d.%d : %s", time.tv_sec…)

QUERY 1: SELECT c.branch FROM CD c, Output o WHERE c.location= o.location and o.value="No electable.." and o.host=1
QUERY 2 : SELECT d.definition FROM DD d WHERE d.use = id_368 and d.variable="send_vote" …

<rep_elect.c>
351 send_vote = rep->winner;

QUERY 3 :SELECT d.definition FROM DD d WHERE d.use=id_351 and d.variable="rep->winner" …

981 if (priority != 0 || LF_ISSET (REPCTL_ELECTABLE)) {
988 } else {
989 rep->winner = DB_EID_INVALID;

QUERY 4: SELECT c.branch FROM CD c WHERE c.location=id_989 and c.host=1
QUERY 5: SELECT s.value FROM State s WHERE s.location=id_981 and s.variable="priority"

QUERY 6 : SELECT d.definition FROM DD d WHERE d.use=id_981 and d.variable="priority" …

252 if (rep->flags & (REP_F_READY_API |
REP_F_READY_OP | REP_F_RECOVER_LOG)){
257 priority = 0;

QUERY 7: SELECT c.branch FROM CD c WHERE c.location=id_257 and c.host=1
QUERY 8 : SELECT d.definition FROM DD d WHERE d.use=id_252 and d.variable="rep->flags" …

<rep_record.c >
476 switch (rp->rectype) {
873 case REP_UPDATE:
878 ret = __rep_update_setup(env, eid, rp, rec);
<rep_backup.c >
__rep_update_setup(..) {
2097 F_SET(rep, REP_F_RECOVER_LOG);

QUERY 9 : SELECT c.branch FROM CD c WHERE c.location=id_2097 and c.host=1
QUERY 10 : SELECT d.definition FROM DD d WHERE d.use=id_476 and d.host=1

<rep_auto.c>
129 DB_NTOHL_COPYIN(env, argp->rectype, bp);

QUERY 11 : SELECT d.definition FROM DD d WHERE d.use=id_129 and DD.variable="bp" and DD.host=1

<repmgr_posix.c>
521 … readv(fd, iovec, buf_count)…

QUERY 12 : SELECT COM.sender, COM.send_point FROM COM WHERE COM.receiver=1 and COM.recv_point=id_521

Sender is NODE 0

2112 (void)__rep_send_message(env, eid, …)
<repmgr_posix.c>
503 … writev(fd, iovec, buf_count)…

QUERY 13: SELECT c.branch FROM CD c WHERE c.location=id_503 and c.host=0

<rep_record.c >
476 switch (rp->rectype) {
601 case REP_LOG_REQ:
… __rep_send_message(env, REP_LOG_REG…)
NODE 0 is down!
447 (void)__rep_send_message(env,
448 eid, REP_LOG, &lsn, &data_dbt,
REPCTL_RESEND, 0);

Related Works

- Queries languages for single process

  [S.Goldsmith et al. '05], [M.C.Martin et al. '05]

- Debugging distributed systems using model checking

  [C.E.Killian et al. '07], [M.Yabandeh et al. '09]

  [M.Musuvathi et al. '04], [J.Yang et al. '06]

- Detect runtime errors in distributed system

  [X.Liu et al. '07], [X.Liu et al. '08]

Related Works

- Debugging based on log mining

  [P.Barham et al. '03], [M.Y.Chen et al. '02],

  [P.Reynolds et al. '06]

- Recording and Replay

  [G.W.Dunalp et al. '02], [D.Geels et al. '07],

  [S.M.Srinivasan et al. '04], [S.Park et al. '09]

- Language based debugging

  [A.Singh et al. '06], [M.Wu et al. '10]

Conclusion

- Recon provides a unified view of distributed system execution for debugging
  - Exposes properties via a relational framework
- Light-weight logging – 3% overhead
- SQL-like query driven replay
  - Populate relations in a demand-driven fashion

Q & A