Scoped Identifiers for Efficient Bit Aligned Logging

Roy Shea[1] Mani Srivastava[2] Young Cho[3]

[1]Department of Computer Science
University of California, Los Angeles

[2]Departments of Electrical Engineering and Computer Science
University of California, Los Angeles

[3]Information Sciences Institute
University of Southern California

March 11, 2010

Overview

- Participated in developing research oriented distributed embedded systems
- Software defects proved very hard to diagnose
  - Standard diagnostic techniques were poor fit
  - Lacked tools to view runtime state of distributed embedded system
- Motivated designing a new logging framework that respects the bandwidth limitations of the distributed embedded domain

Debugging Distributed Systems is Hard

Algorithm Design

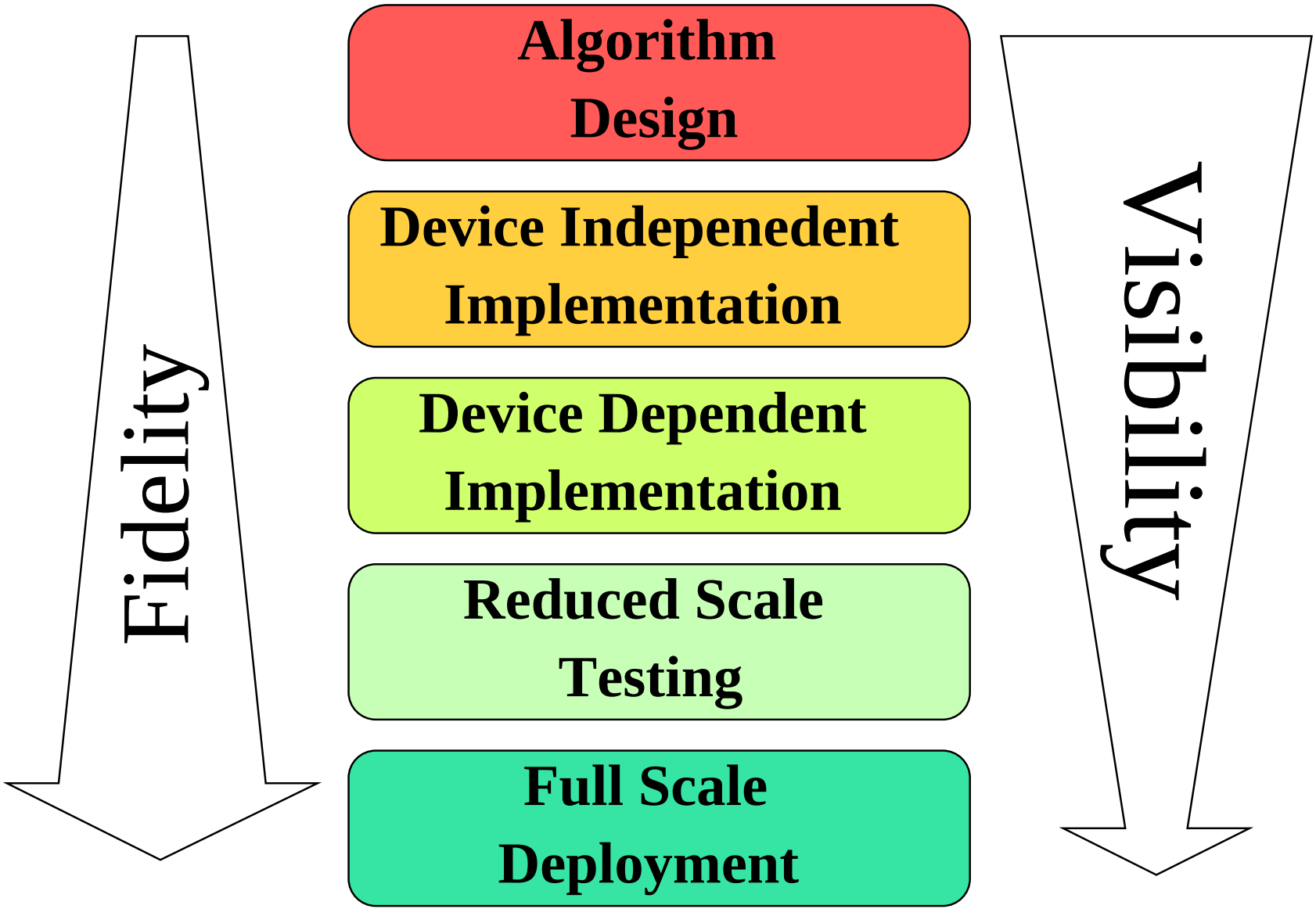
Device Indepenedent Implementation

Device Dependent Implementation

Reduced Scale Testing

Full Scale Deployment

Debugging Distributed Systems is Hard



Why Logging?

Runtime Solution

Because defects emerged during deployment despite good design practices

Passive Solution

Because interactive techniques have limited applicability to domain

Runtime logs explain unexpected system behavior without user interaction during deployment

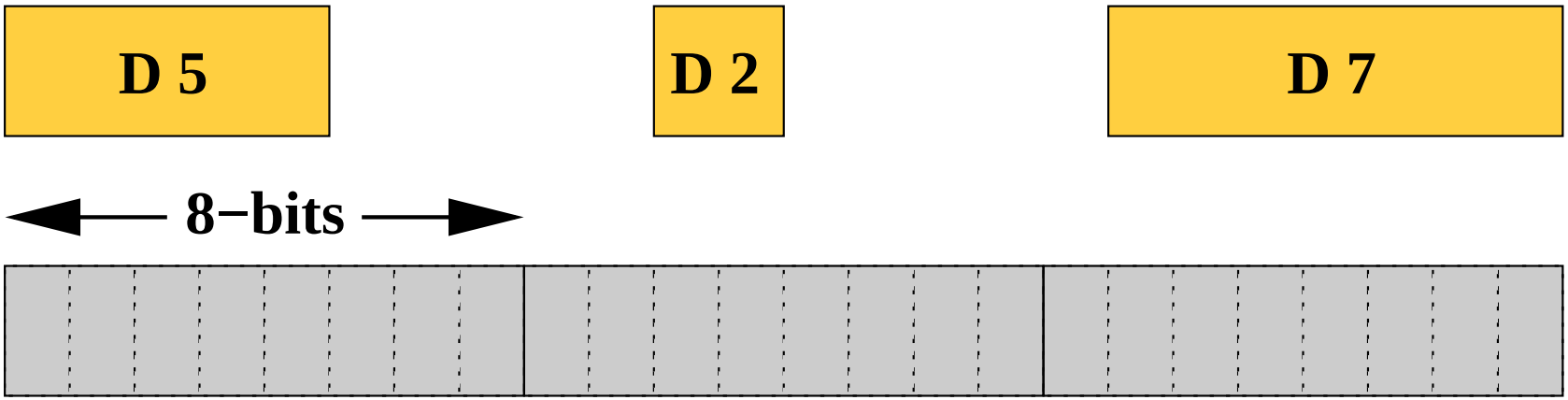
Growing Beyond `printf`

- Logging has a long history of `printf` style interfaces
  - Immediately understandable to developer
  - Easy to integrate into regular work flow

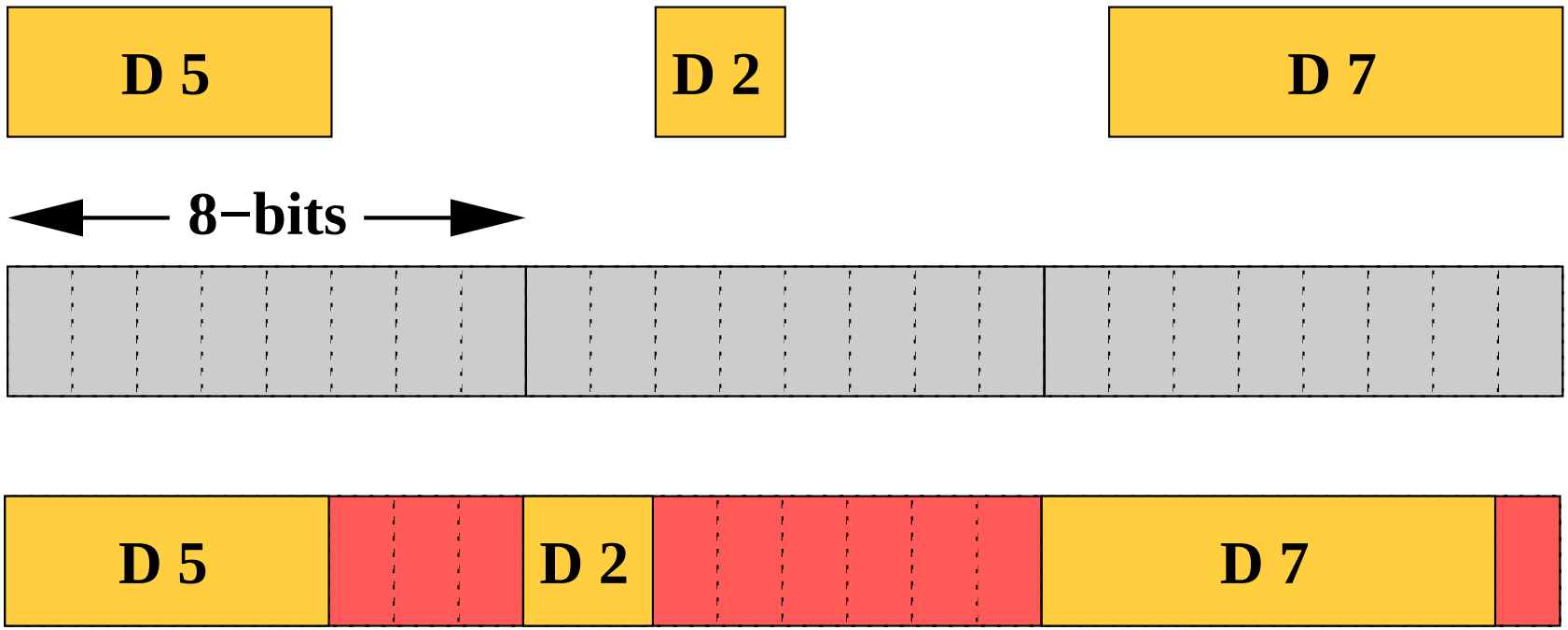
Growing Beyond `printf`

- Logging has a long history of `printf` style interfaces
  - Immediately understandable to developer
  - Easy to integrate into regular work flow
- Verbose free form logging stresses bandwidth in distributed embedded systems
- External research re-encodes ASCII strings with numeric identifiers to reduced log bandwidth

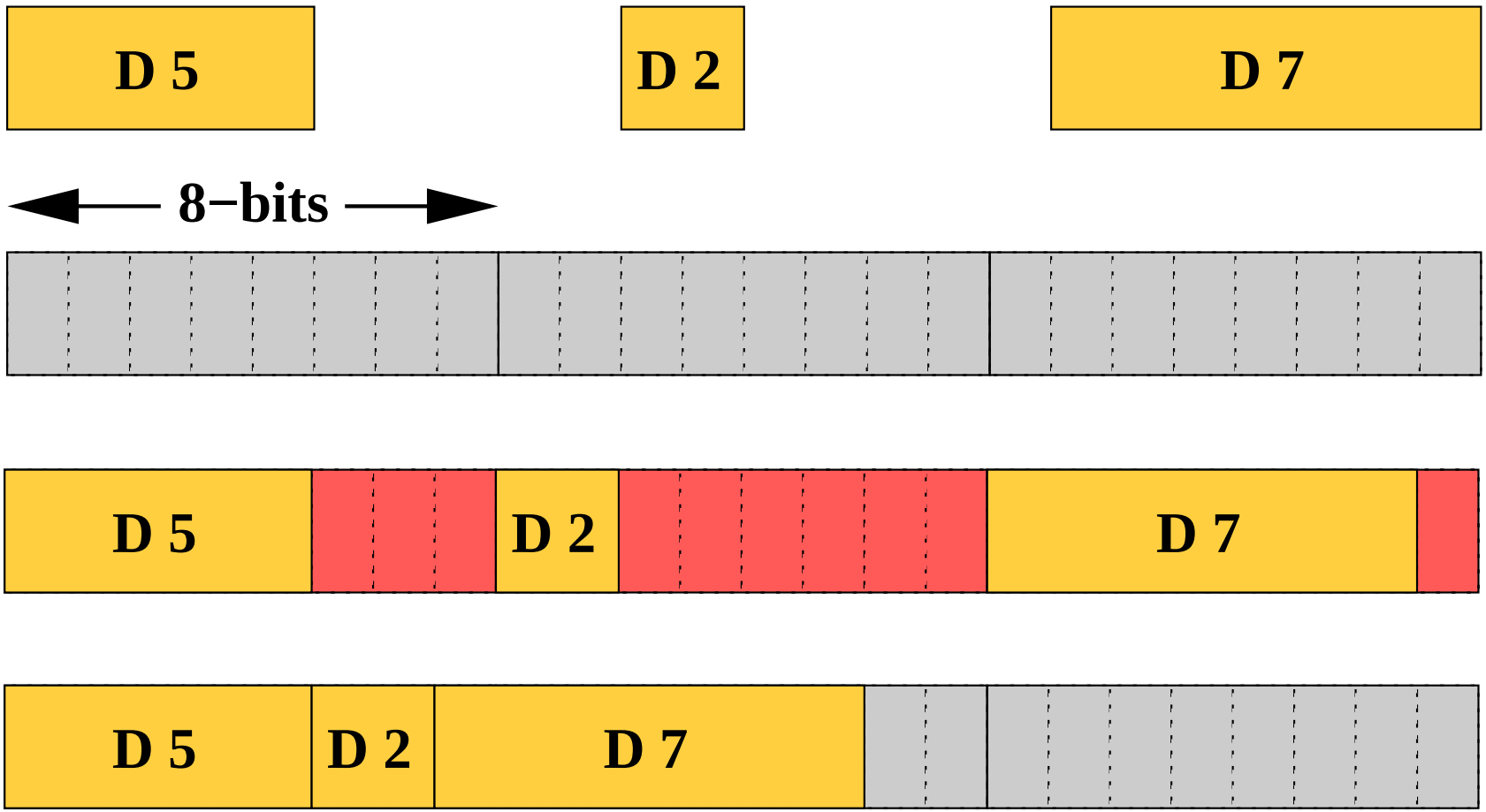
Savings From Bit Aligned Logging



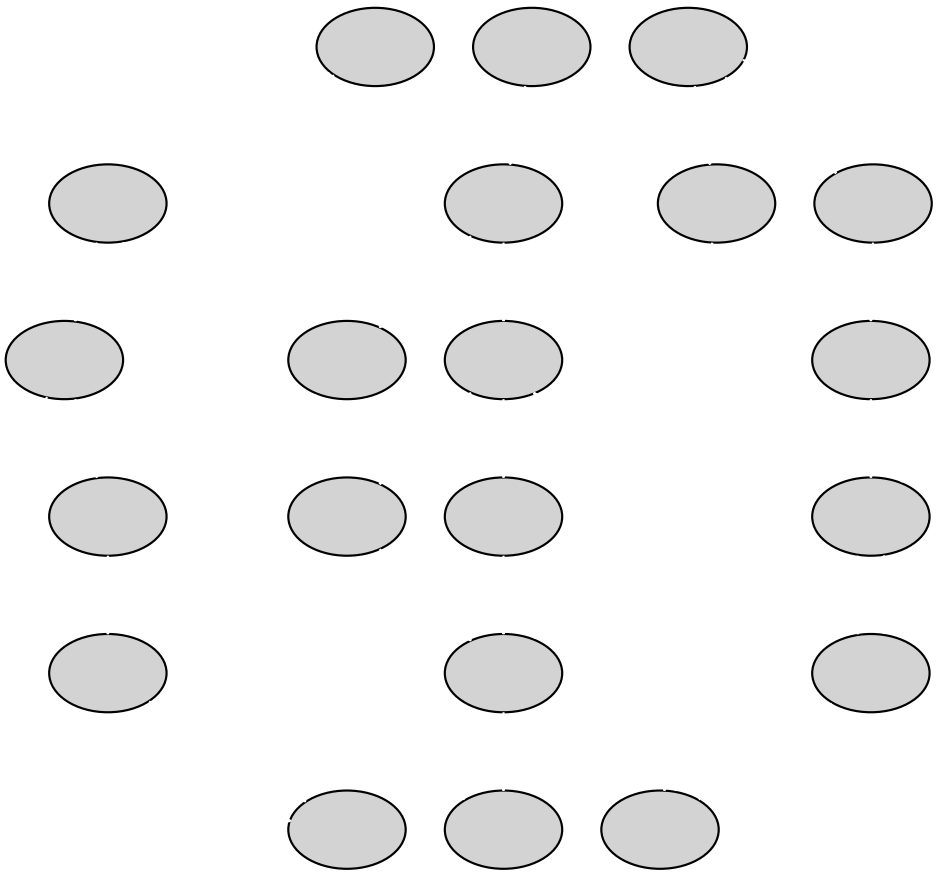
Savings From Bit Aligned Logging



Savings From Bit Aligned Logging



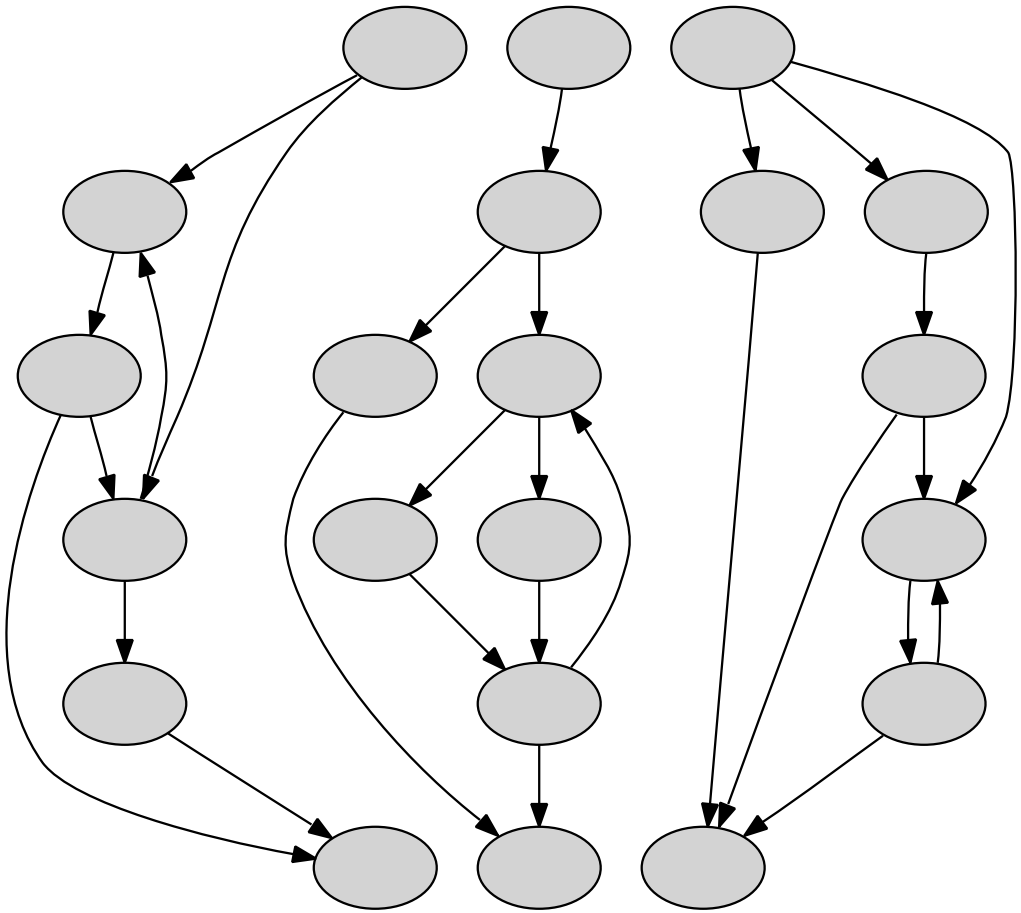
Standard Identifier Assignment



Identifiers are typically assigned from a global name space

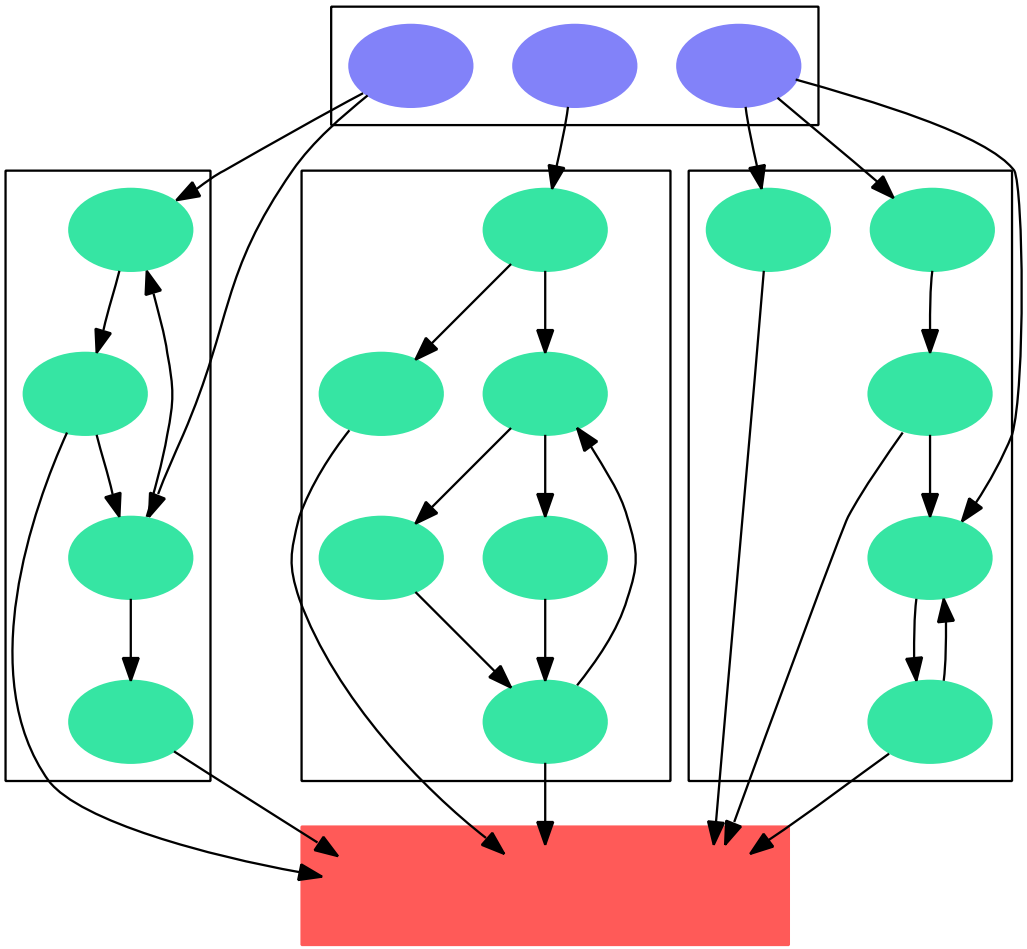
21 tokens in a single name space requires* 5-bits to encode

Real Programs Have Rich Structure



Exploit program structure to more compactly encode tokens

Local Token Scoping for Short Identifiers



3 tokens → 2 bits

6 tokens → 3 bits

1 token → 1 bit

Log Instrumentation Specifications (LIS)

- Scripting language to describe logging tasks
- Generalized logging solution providing explicit scoping declarations

Global	tokens assigned identifiers unique across all tokens
Local	tokens assigned identifiers unique within a function
Point	token uses a single well defined identifier

LIS Scripts

- LIS scripts reside outside of the code base
- Script is composed of statements describing runtime state to be gathered
  - Base statement types target key function properties
  - Combination of statement type, function name, and optional modifiers control location of instrumentation
- LIS scripts are written by developers or emitted by higher level analysis

Demonstrative Logging Task

```
void read_done(error_t result, uint16_t data) {

    if (send_busy == TRUE) {


        return;
    }

    /* Rest of function body elided... */

    return;
}
```

Demonstrative Logging Task

```
header read_done global
```

```
void read_done(error_t result, uint16_t data) {
    bitlog_write(4, 3); /* Log header */
    if (send_busy == TRUE) {


        return;
    }

    /* Rest of function body elided... */

    return;
}
```

Demonstrative Logging Task

```
header read_done global
```

```
controlflow read_done local if send_busy
```

```
void read_done(error_t result, uint16_t data) {
    bitlog_write(4, 3); /* Log header */
    if (send_busy == TRUE) {
        bitlog_write(5, 3); /* Log control flow */

        return;
    }
    bitlog_write(6, 3); /* Log control flow */

    /* Rest of function body elided... */

    return;
}
```

Demonstrative Logging Task

```
header read_done global
controlflow read_done local if send_busy
footer read_done point
```

```
void read_done(error_t result, uint16_t data) {
    bitlog_write(4, 3); /* Log header */
    if (send_busy == TRUE) {
        bitlog_write(5, 3); /* Log control flow */
        bitlog_write(0, 1); /* Log footer */
        return;
    }
    bitlog_write(6, 3); /* Log control flow */
    /* Rest of function body elided... */
    bitlog_write(0, 1); /* Log footer */
    return;
}
```

Demonstrative Logging Task

```
header read_done global
controlflow read_done local if send_busy
footer read_done point
```

```
void read_done(error_t result, uint16_t data) {
    bitlog_write(4, 3); /* Log header */
    if (send_busy == TRUE) {
        bitlog_write(5, 3); /* Log control flow */
        bitlog_write(0, 1); /* Log footer */
        return;
    }
    bitlog_write(6, 3); /* Log control flow */
    /* Rest of function body elided... */
    bitlog_write(0, 1); /* Log footer */
    return;
}
```

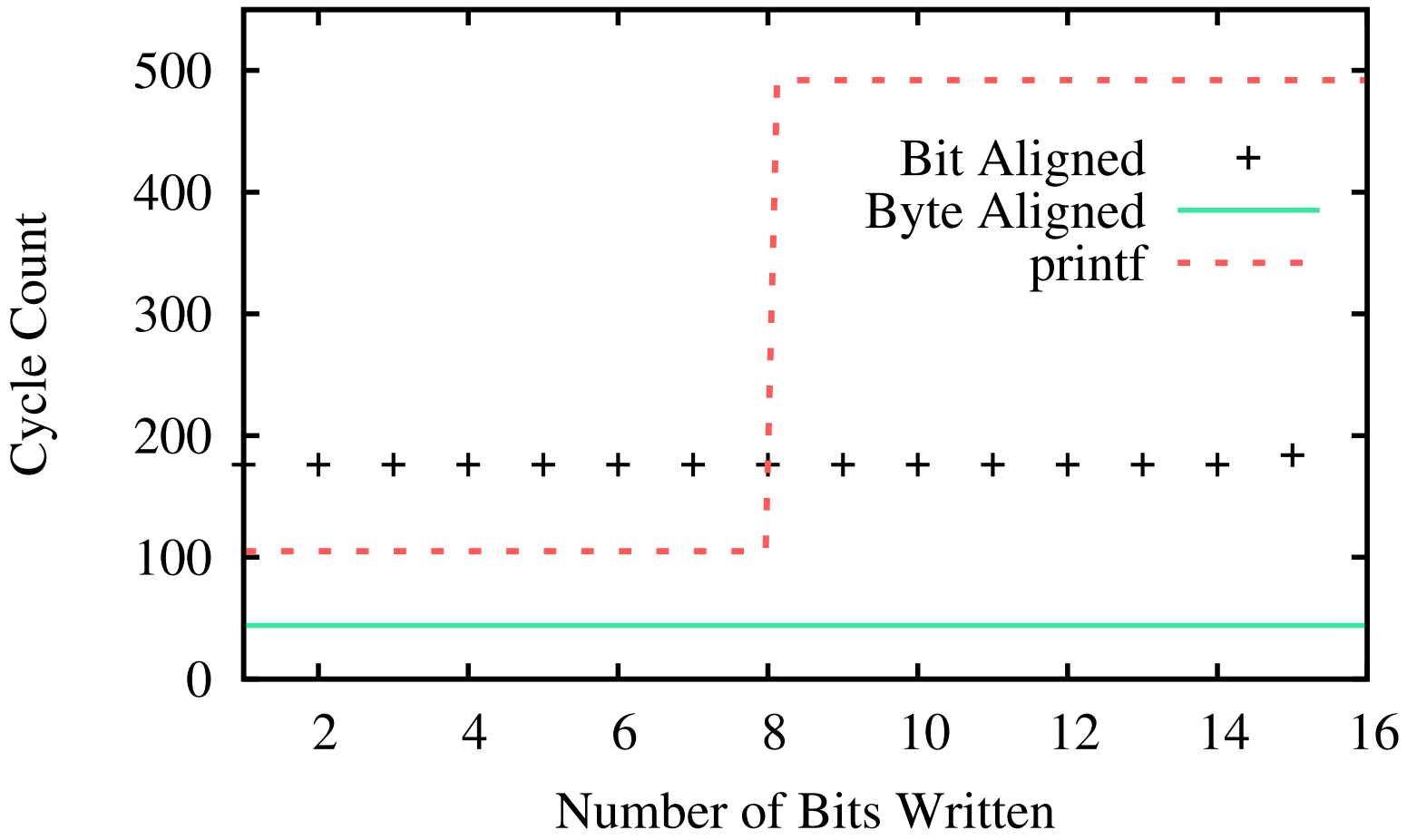
Evaluation Background

- Implemented small stand alone `bitlog` library
- LIS framework developed to instrument C source code
- Evaluation targets the ATMega128 microcontroller and CC2420 radio
- Cycle counts obtained using the cycle accurate Avrora simulator
- Bandwidth measurements obtained using small testbed

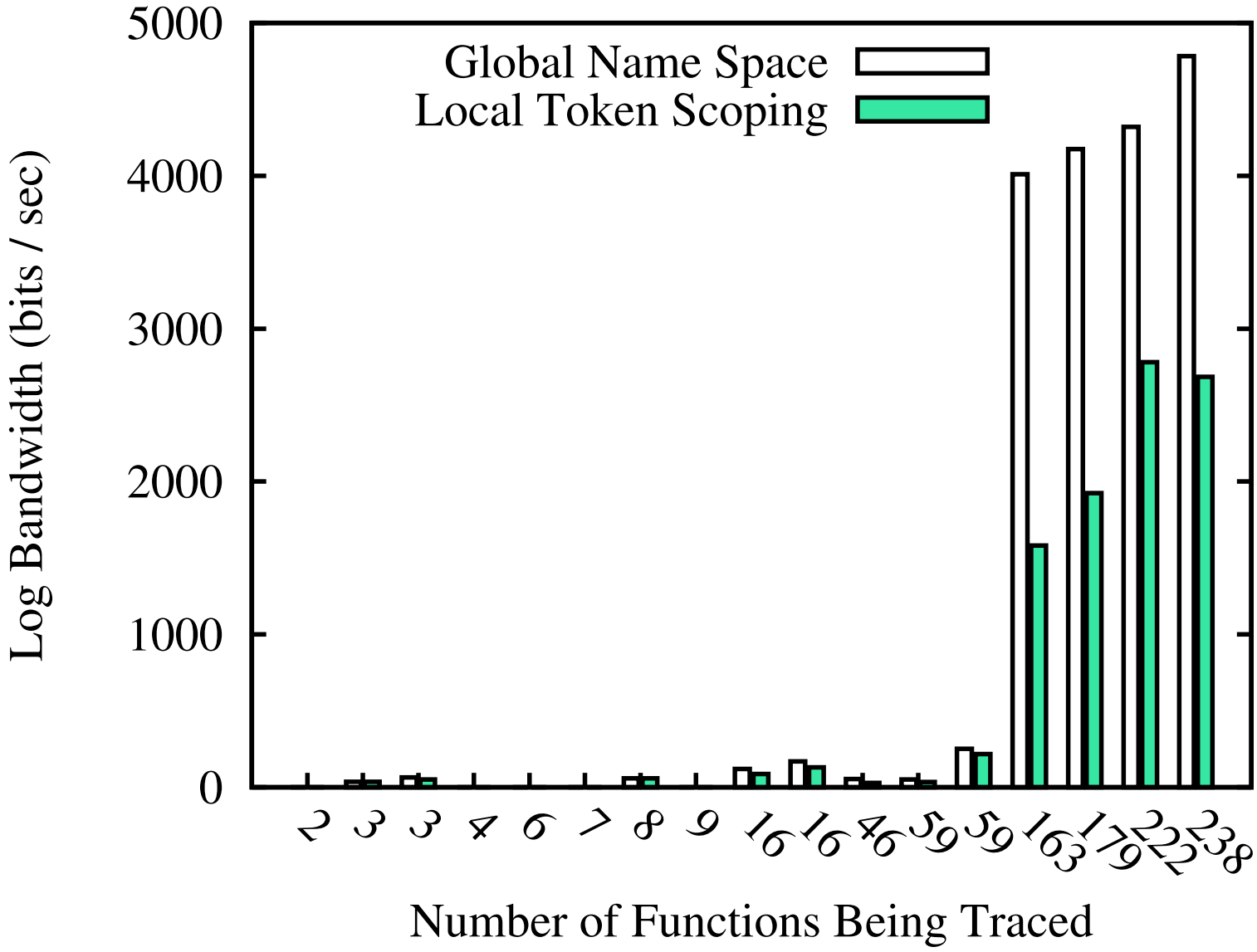
Static Costs of Bit Aligned Logging

System Component	Program Memory (bytes)	Data Memory (bytes)
Radio Stack	9264	210
Routing Layer	10284	1360
Routed Log Management	1412	351
Broadcast Log Management	74	128
Bitlog Library	290	24
Call to `bitlog_write`	14	0

Latency from Bit Aligned Logging



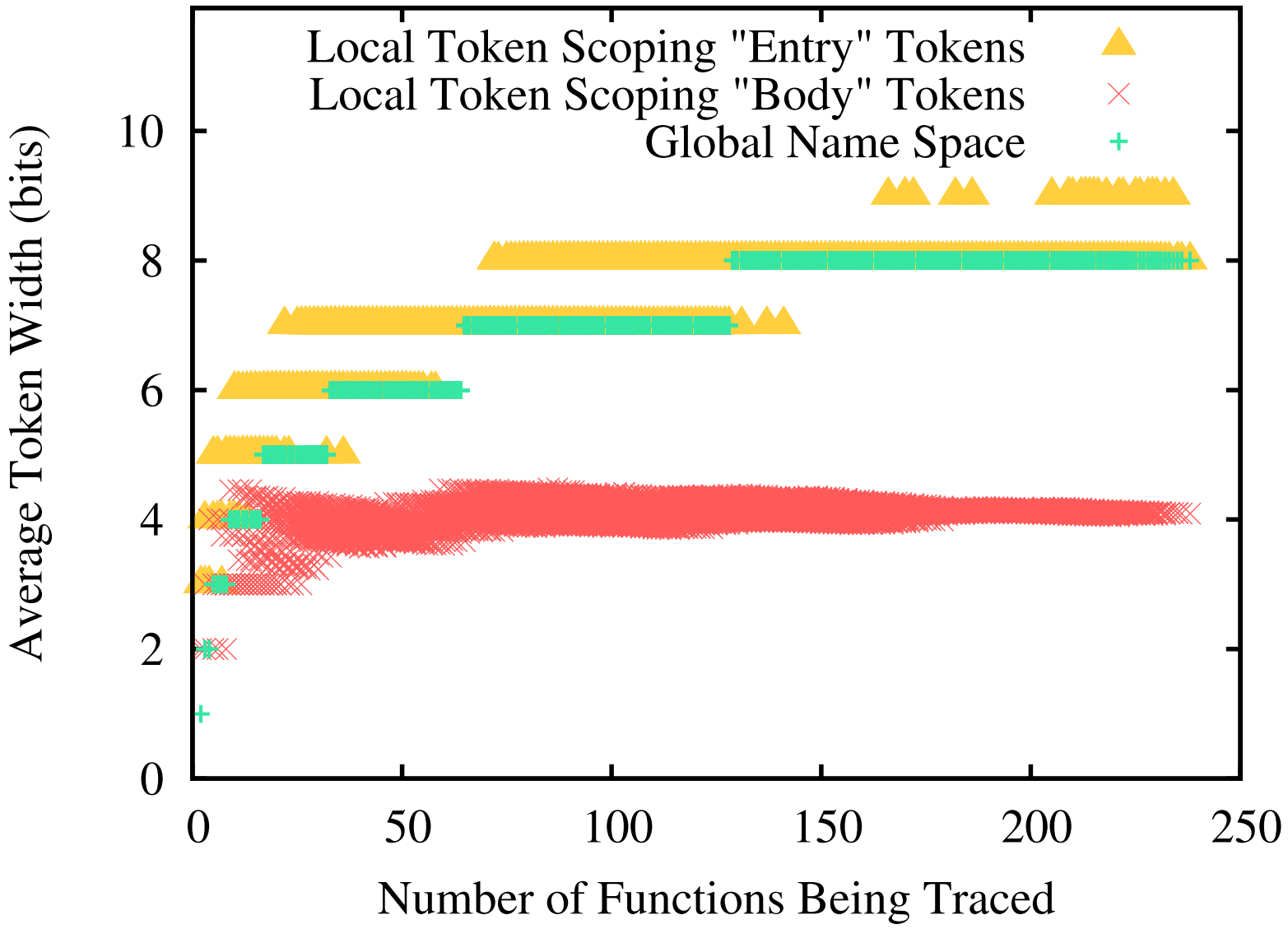
Number of Tokens Affects Bandwidth



Observed Run Time Identifier Size

		Average Size (bits)	
Alignment	**Scoping**	**$<$ 60 Tokens**	**$>$ 150 Tokens**
Byte	—	8.0 ± 0.0	8.0 ± 0.0
Bit	Global	3.5 ± 1.6	8.0 ± 0.0
Bit	Local	2.8 ± 0.5	3.7 ± 0.2

Number of Tokens Affects Identifier Width



Limitations

- Current LIS implementation bases scopes on function boundaries
- Token scoping does not compress program data
- Care must be taken to guarantee parsable logs
  - Parsing depends on knowing the local context
  - Script writing guidelines guarantee parsable logs

Conclusions

- Established technique of bit aligning data used to reduce log bandwidth...
  - but only effective with consistently small identifiers
- Local *token scoping* can produce consistently small token identifier encodings...
  - but requires a great deal of error prone book keeping
- LIS automates the difficulties of token scoping and provides a general interface for describing logging tasks

Questions?

https://projects.nesl.ucla.edu/~rshea/lis/