Knit: Component Composition for Systems Software

Alastair Reid, Matthew Flatt, Leigh Stoller, Jay Lepreau, Eric Eide

University of Utah

Why Components?

- Everyone is writing too much code
  - Not enough code reuse
  - Hard to reconfigure
  - Hard to understand
  - Hard to test/verify
- Exceptions: Click, Scout, Ensemble, Fox, MMLite, OSKit, …

Why Not Components?

- Overhead
  - Runtime
  - Programmer time
- Advanced systems don’t work with C
- Complex component interdependencies
  - Locking restrictions
  - Top/bottom-half
  - Bootstrap sequence

Goal of Knit Project

To make components practical for systems programming

Key to Achieving Goal

Static configuration language

- Enables error detection
- Enables optimization

Target#1: The Utah OSKit [SOSP'97]

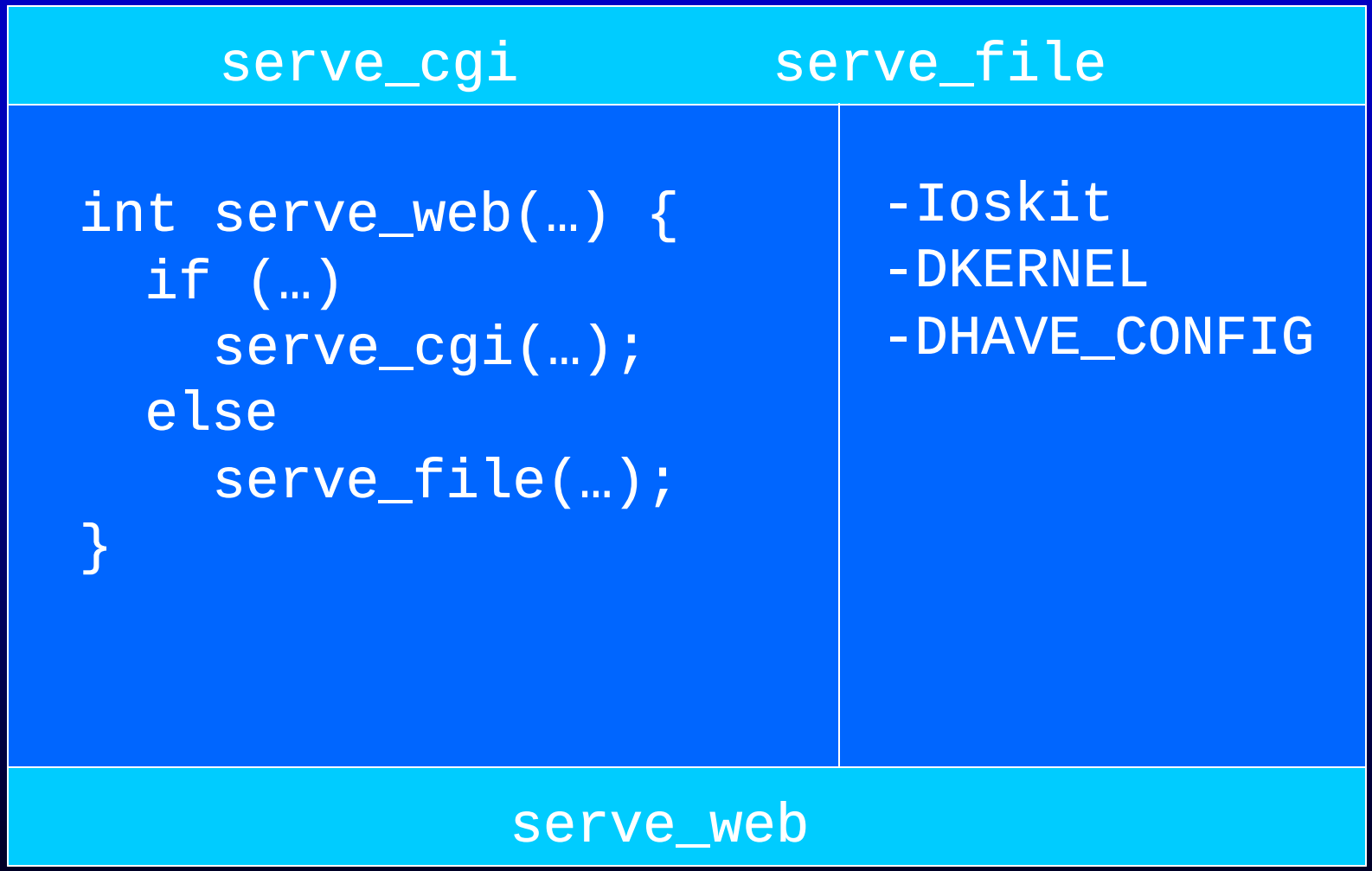
- Approximately 500 components:

  Device drivers, bootstrap code, TCP/IP stacks, filesystems, SNMP, etc.
- Doesn't impose architecture
- 10^6 lines of code from Linux, FreeBSD, NetBSD, Mach, Fluke, etc.

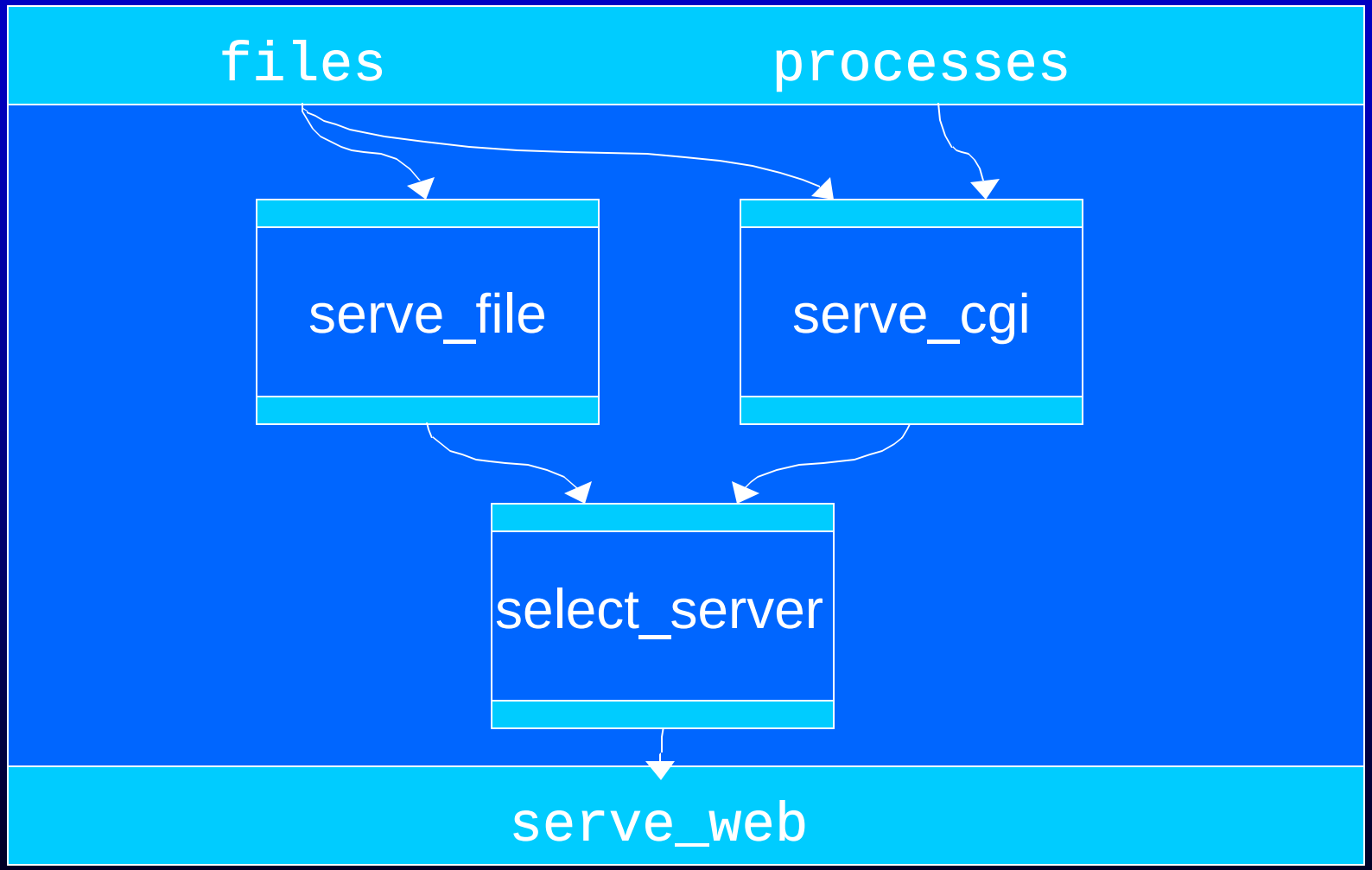
Outline

- Introduction
- The Knit component model
  - Atomic units
  - Compound units
  - Automatic Initialization
  - Detecting Configuration Errors
- Implementation and Performance
- Open issues

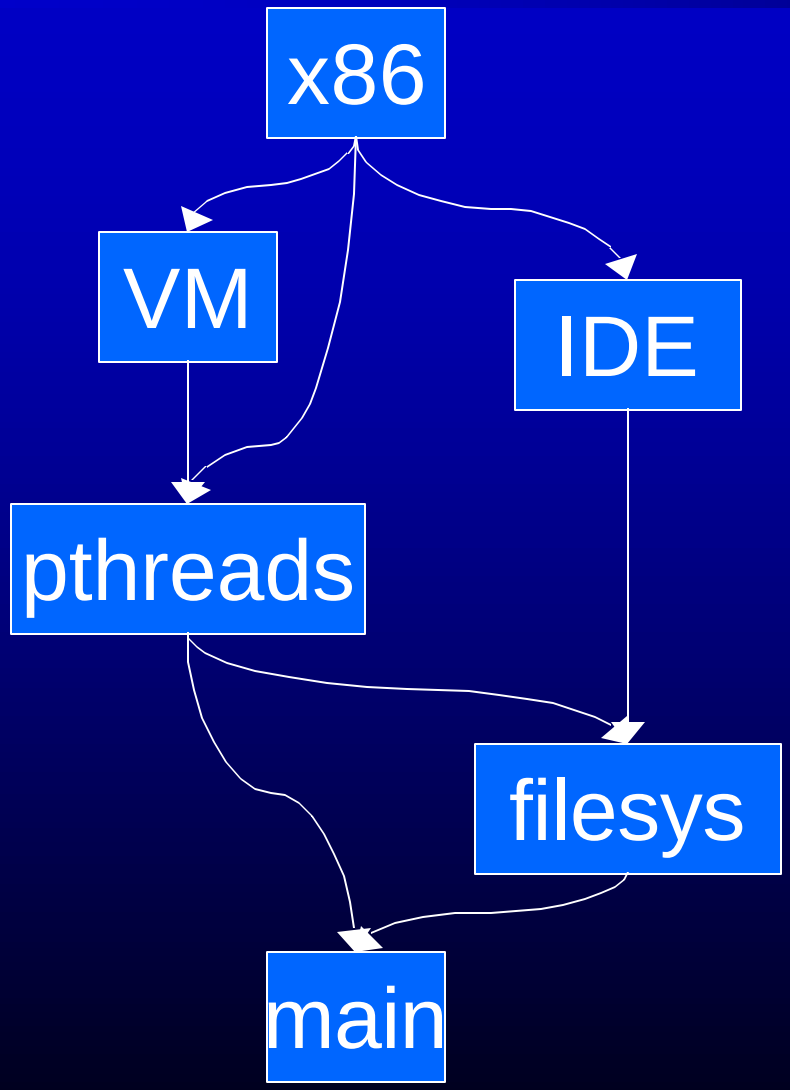
Atomic Units [PLDI'98]



Compound Units [PLDI'98]

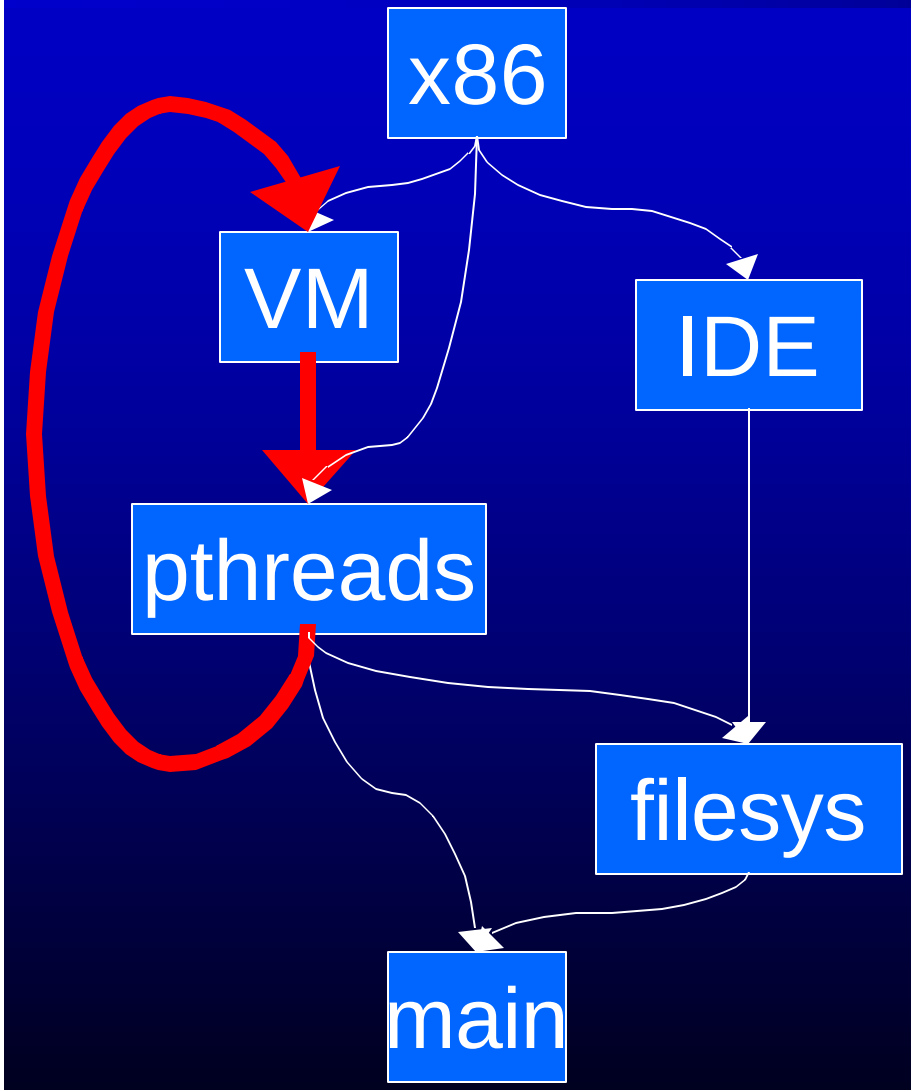


Initialization




```
init_x86();
init_IDE();
init_VM();
init_threads();
init_filesys();
init_main();
```

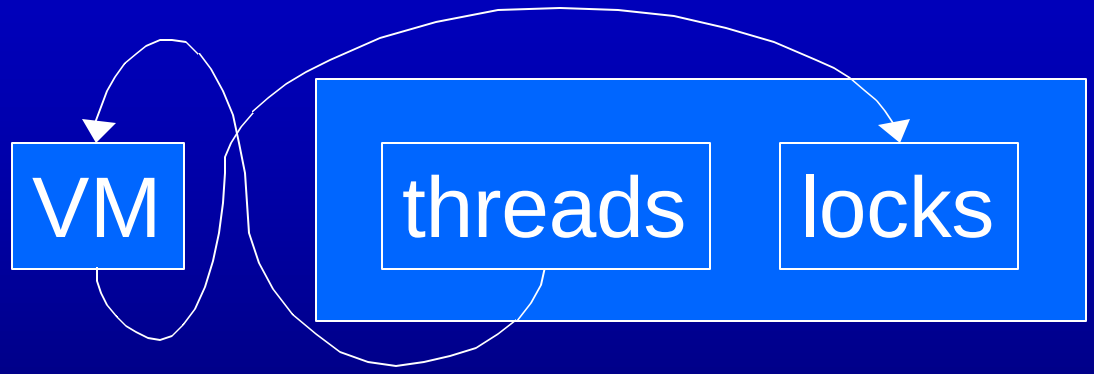
Initialization



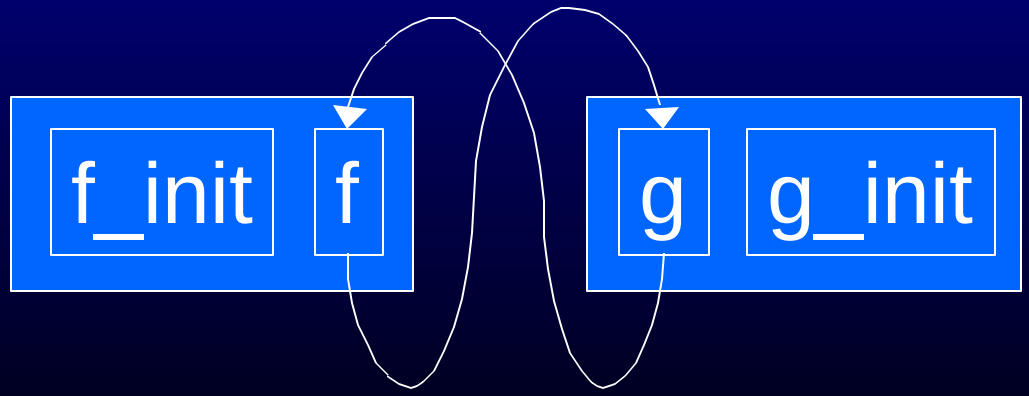
```
init_x86();
init_IDE();
init_VM();
init_pthreads();
init_filesys();
init_main();
```

When Can We Break Cycles?

1. Component ‘contains’ subcomponents



2. No dependency between initializers

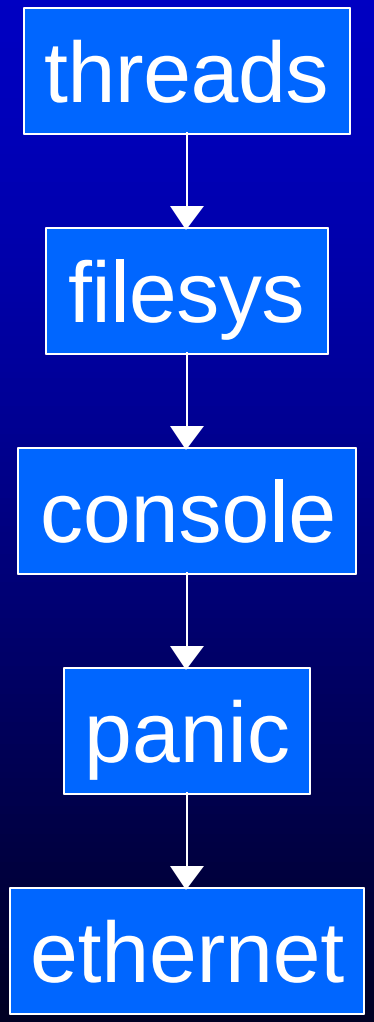


Automatic Initialization

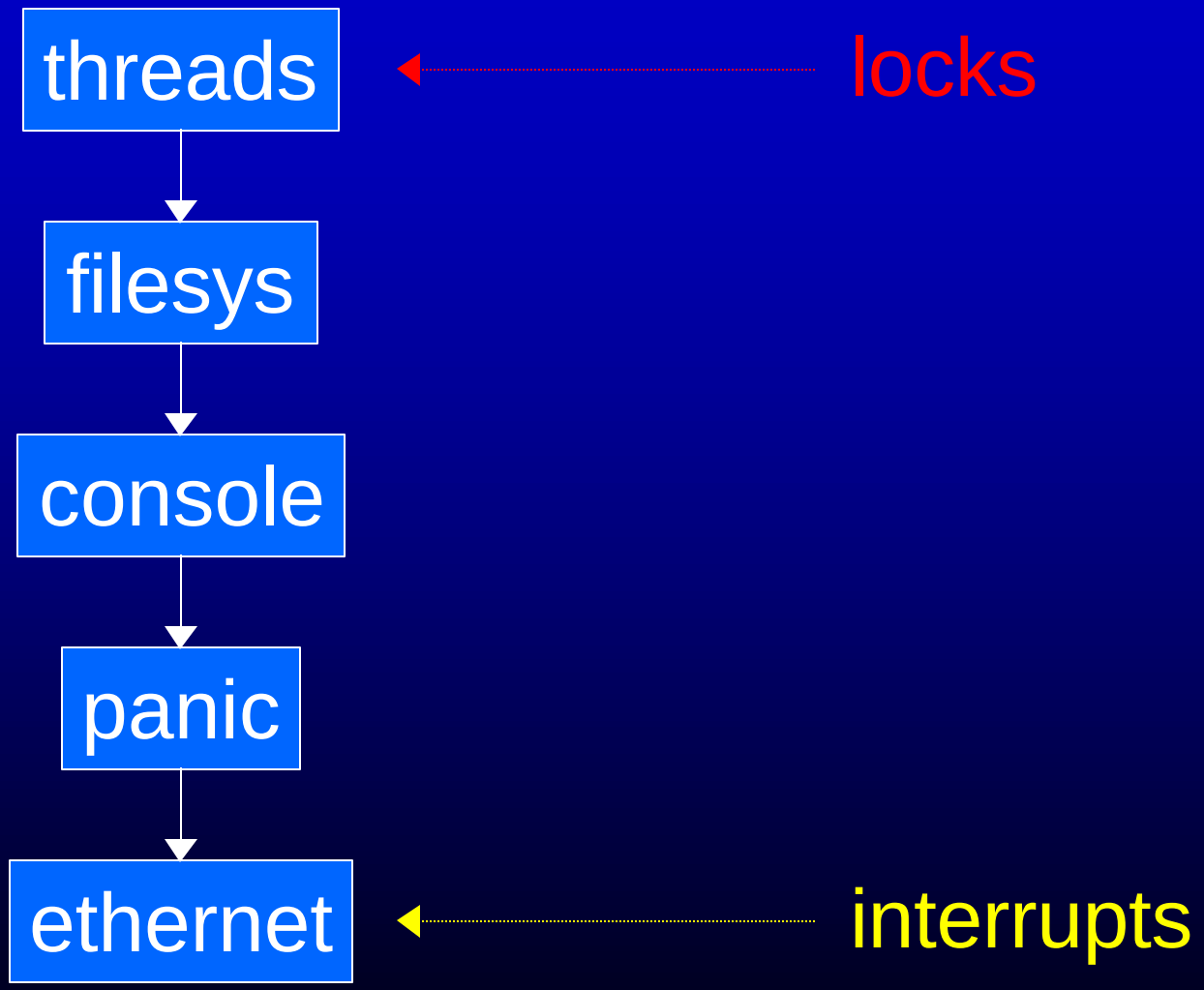
- Knit generates initialization sequence
- Cycles are resolved by refining initialization dependencies in units

- Experience
  - 5% of units need dependencies refined
  - Programmers find initialization a big win

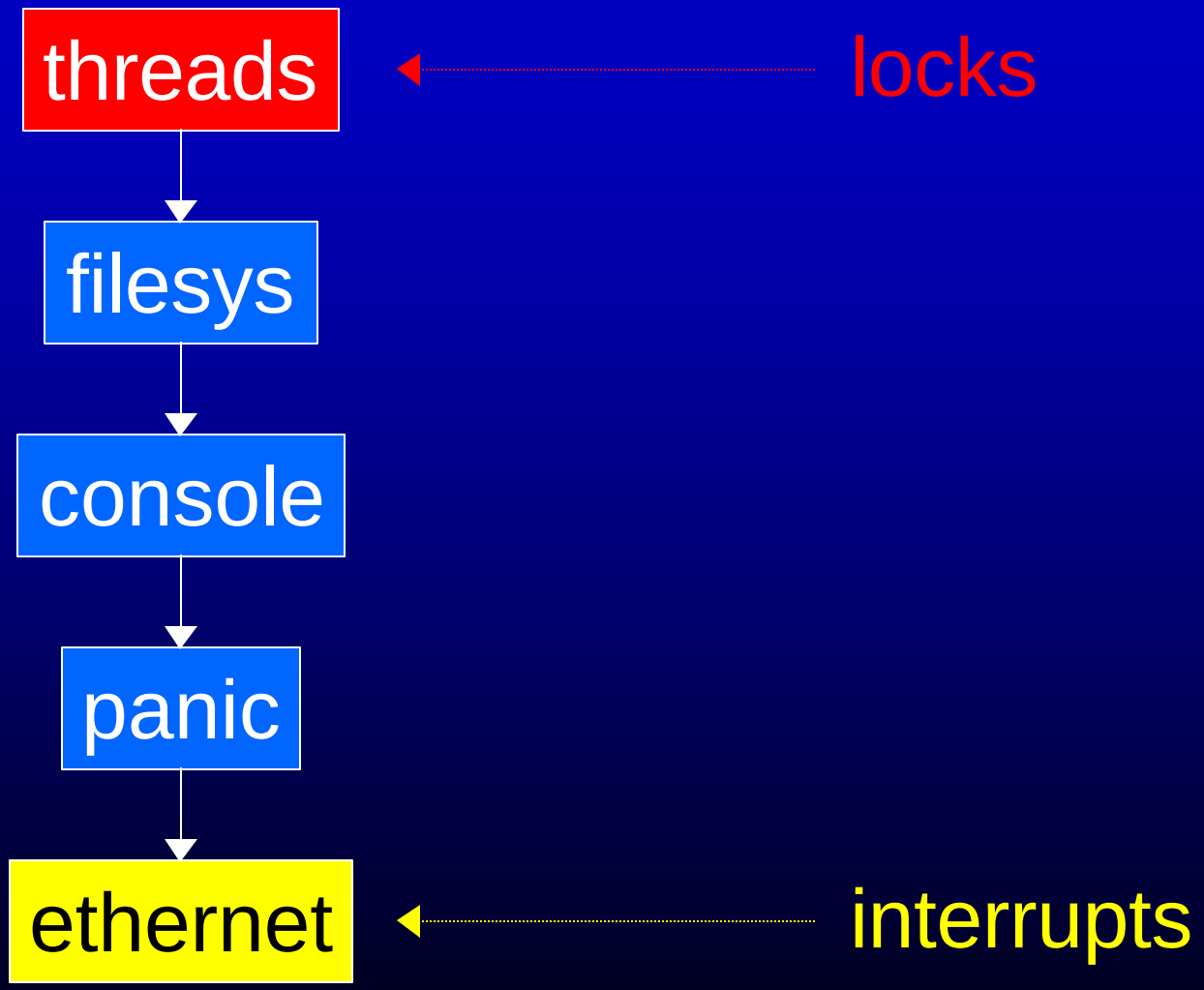
Detecting Composition Errors



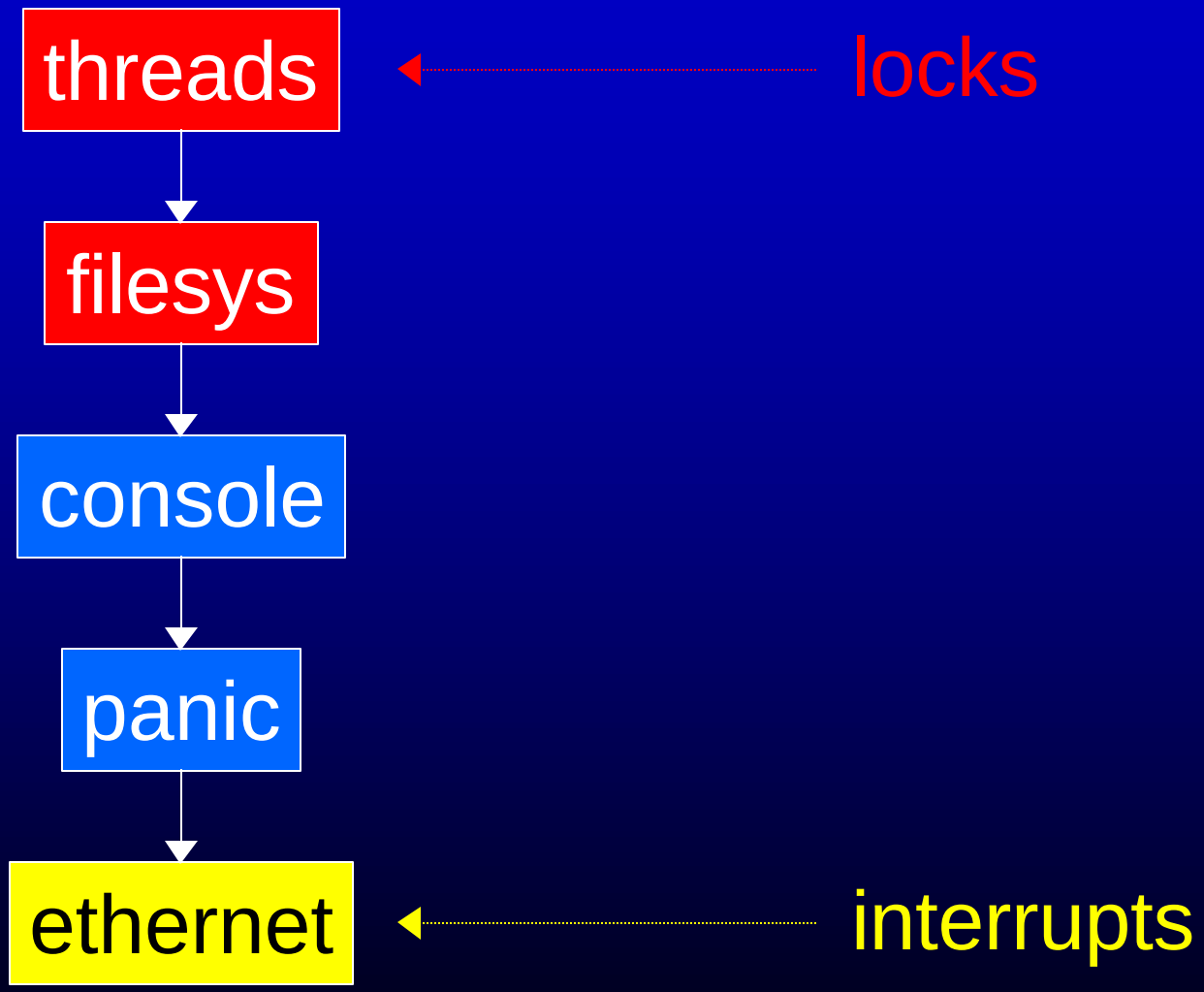
Detecting Composition Errors



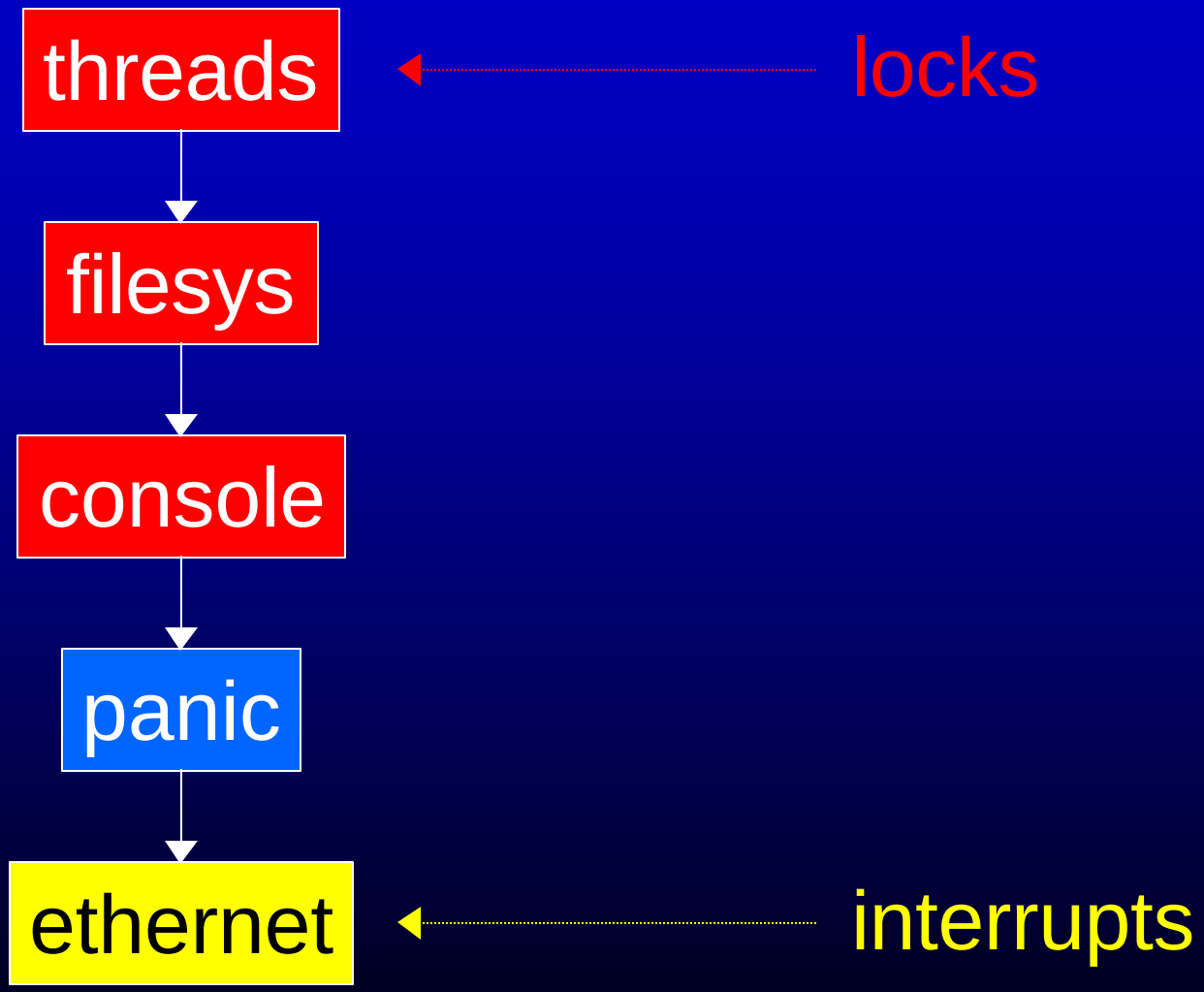
Detecting Composition Errors



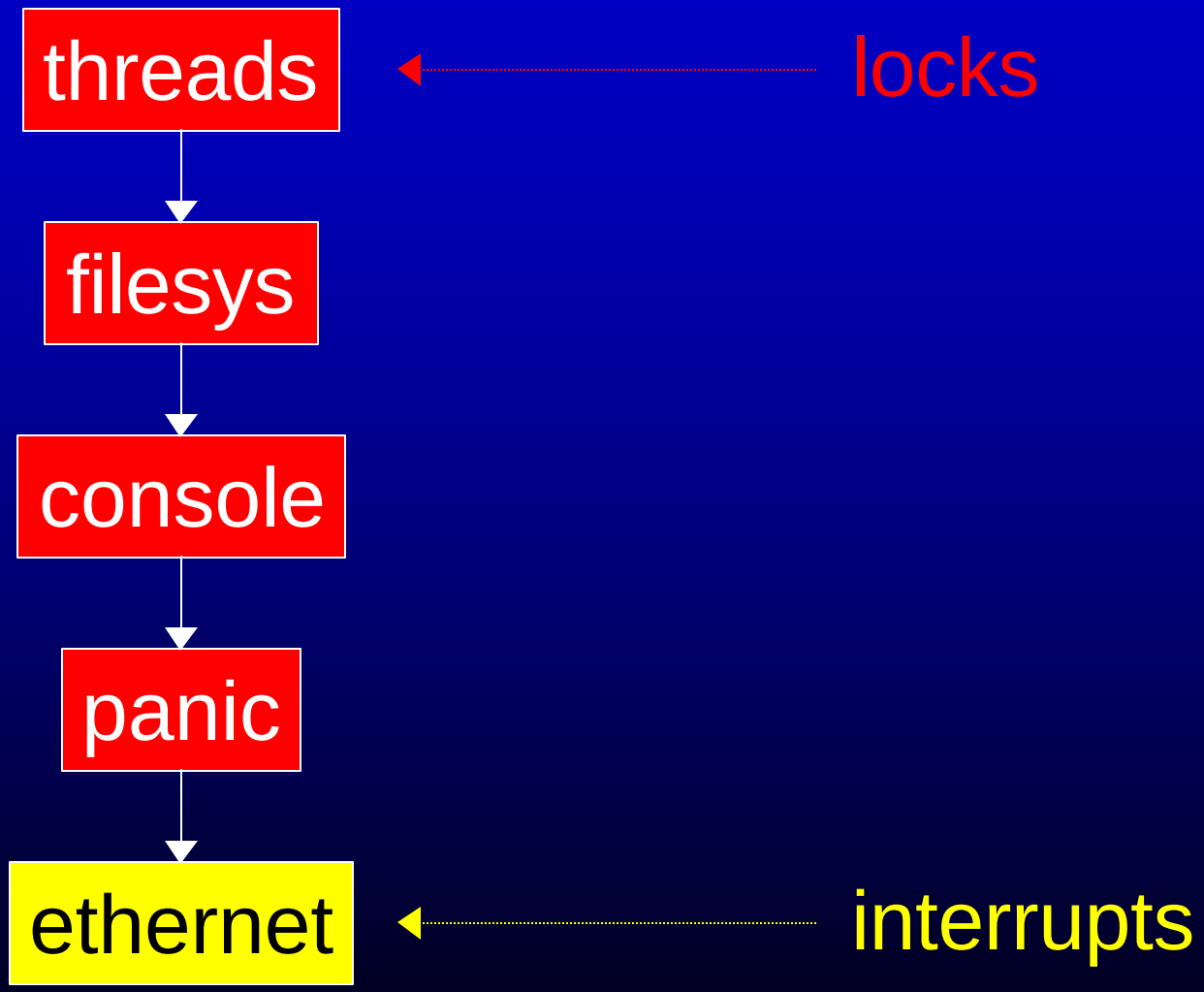
Detecting Composition Errors



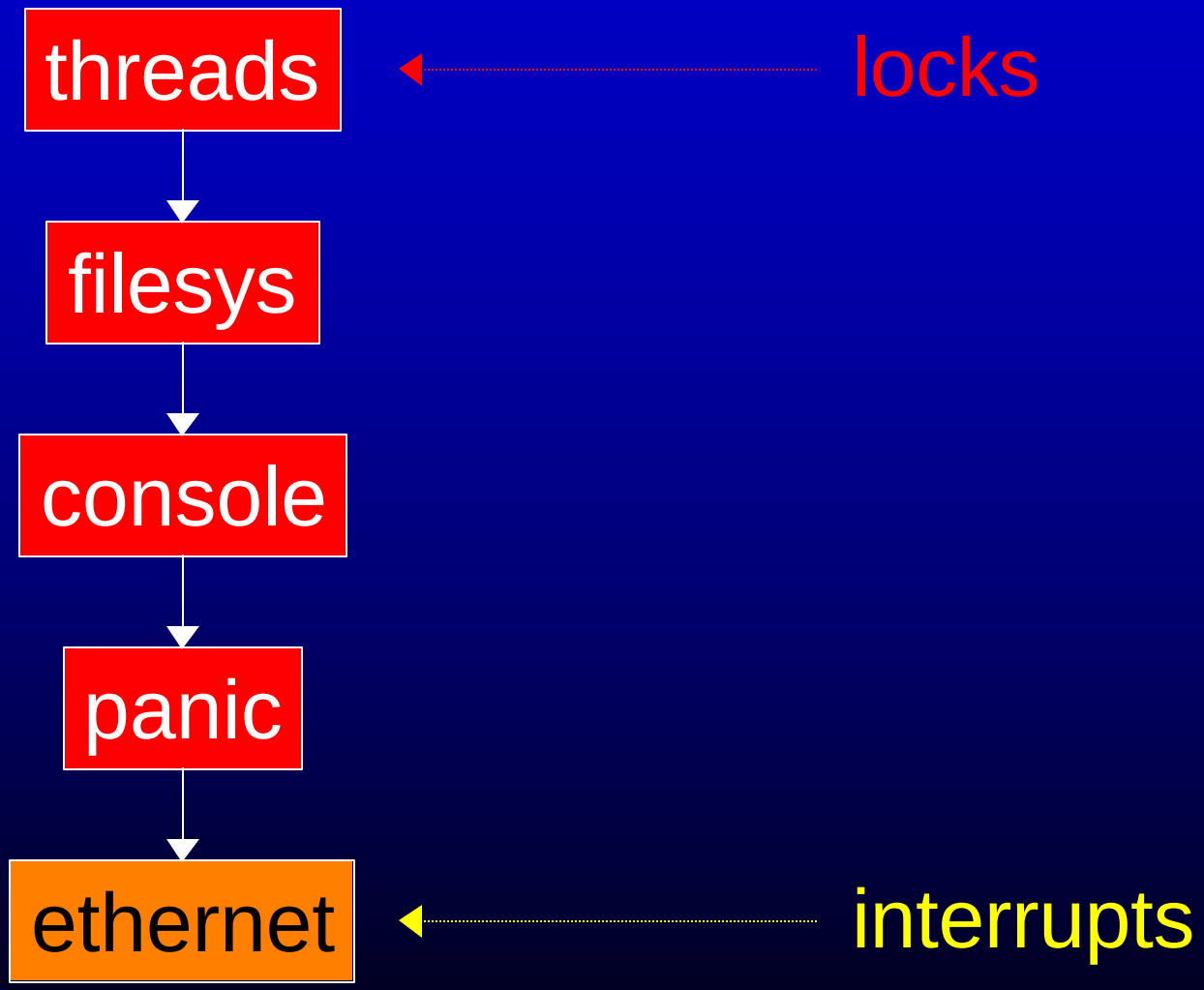
Detecting Composition Errors



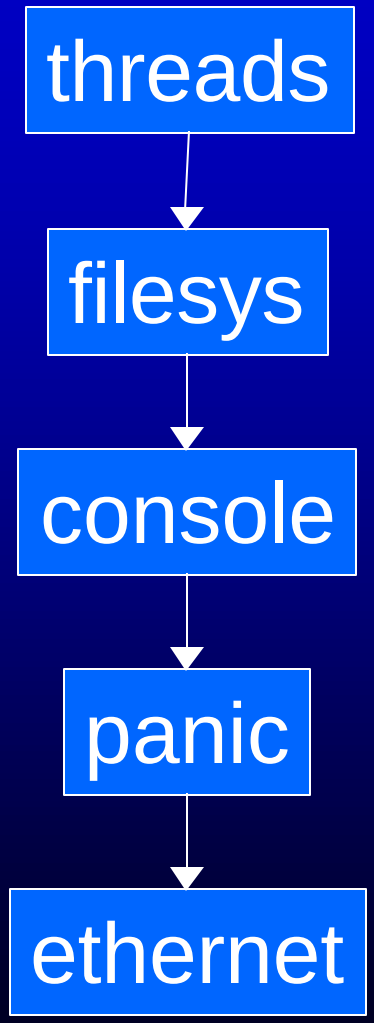
Detecting Composition Errors



Detecting Composition Errors



Detecting Composition Errors



context(threads) <= ProcessContext

context(filesys) <= context(threads)

context(console) <= context(filesys)

context(panic) <= context(console)

NoContext <= context(ethernet)

ProcessContext < NoContext

Extensible Constraint System

- Constraint system propagates properties through component interconnections
  - Knit can detect global errors
- Constraint system is extensible
  - In context X, don’t do Y
  - Type system for Modular IP Routers (e.g., Click)
  - …

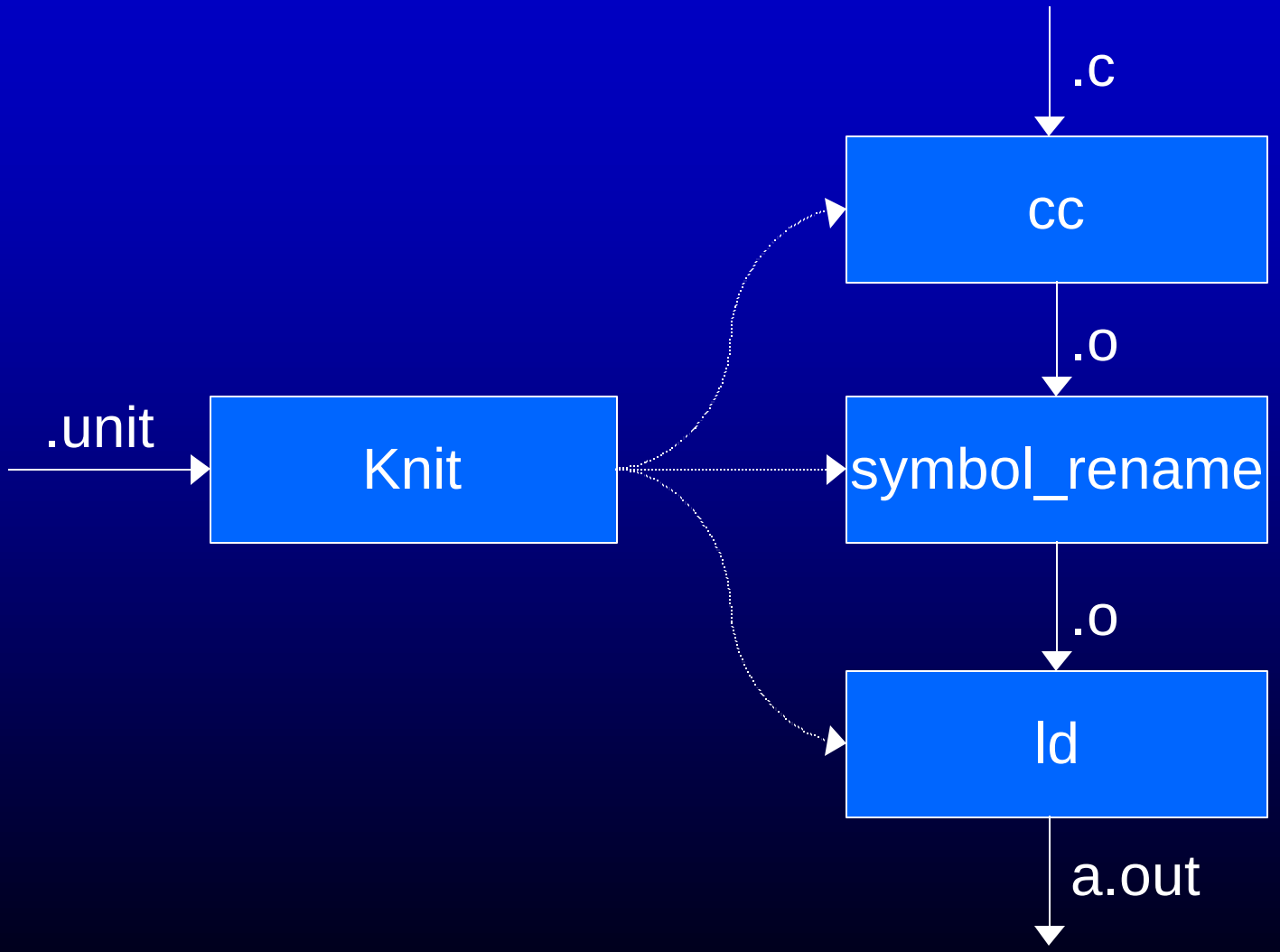
Knit

- Supports C, assembly and object files
- Separates interconnections from code
- Automatic initialization
- Extensible constraint system
- Allows cyclic component dependencies
- Allows multiple instances of components
- Text based

Outline

- Introduction
- The Knit component model
- Implementation and Performance
- Open issues

Implementation (Unoptimized)



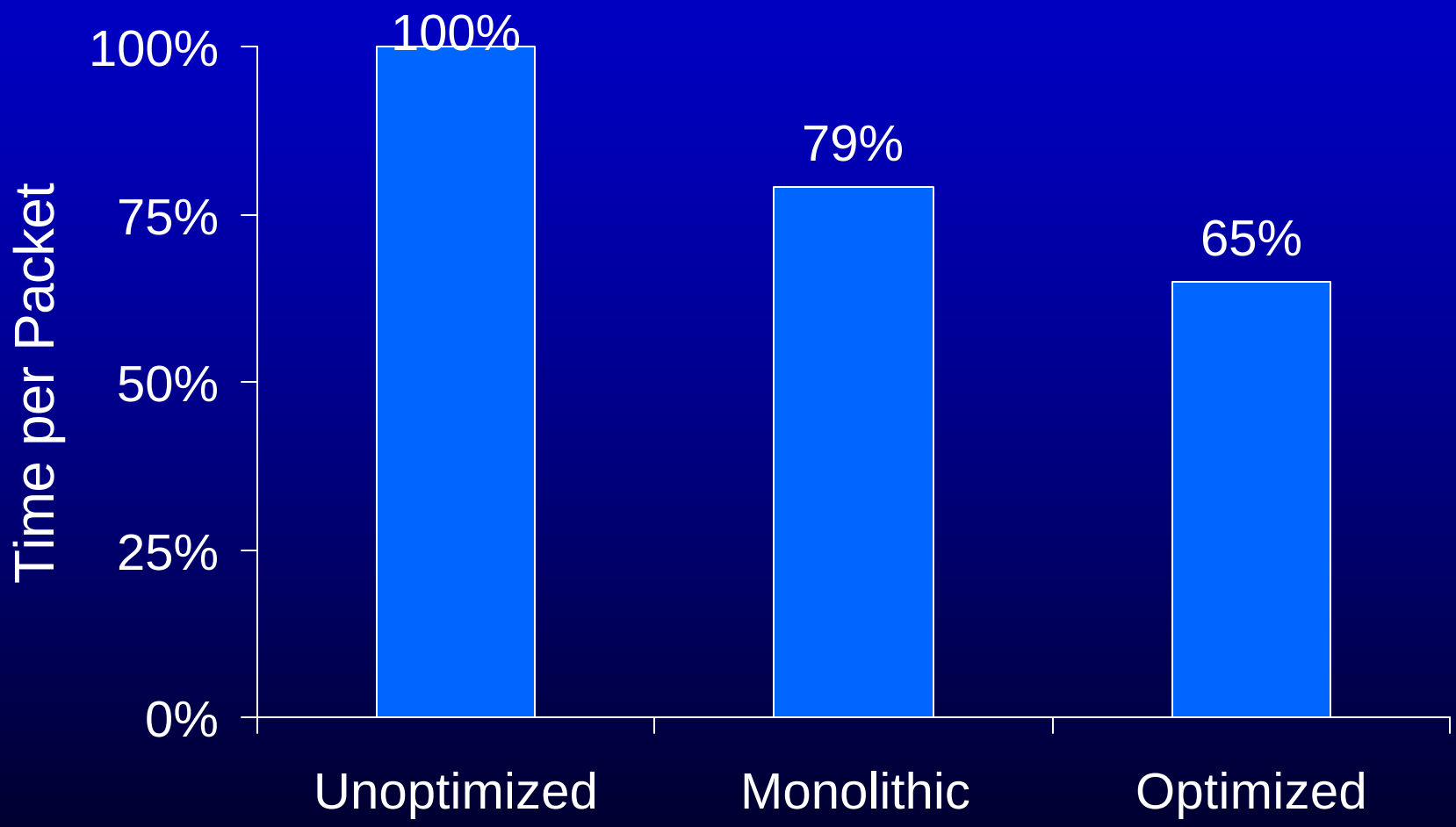
Performance

- Component cost should not distort system structure
- Reduce overhead by eliminating function calls

Click and Clack

- Click modular network router from MIT [SOSP'99]
- Clack
  - Re-implementation of Click using Knit
  - Similar performance to Click
- Many small components

Performance of Clack



Open Issues

- Is Knit general purpose?
  - Need more users
  - Need more applications
- Is the constraint system extensible enough?
- Implicit linking vs. explicit linking?

Conclusions

- State of the art component system for C
- Targeted at systems code
  - Automatic initialization
  - Detects local and global errors
  - Low performance overhead

- Available ASAP: http://www.cs.utah.edu/flux/