Alchemy

Jay Lepreau (PI), Matthew Flatt (PI),
Alastair Reid, Eric Eide, Leigh Stoller,
Mike Hibler

University of Utah

Overview

- Why Components?
- Components and AOP
- Barriers to Component Programming
- Knit: Component linking and definition language with a strong practical bent
- Status and Future Work

What is a component?

Modules with:

- Clearly defined exports
- Clearly defined imports
- Control over component instantiation
- Control over component interconnection

- Source code? No source code?
- Can be distributed across machines?

Why Components?

- Reuse
- Isolation
- Documentation
- Flexibility

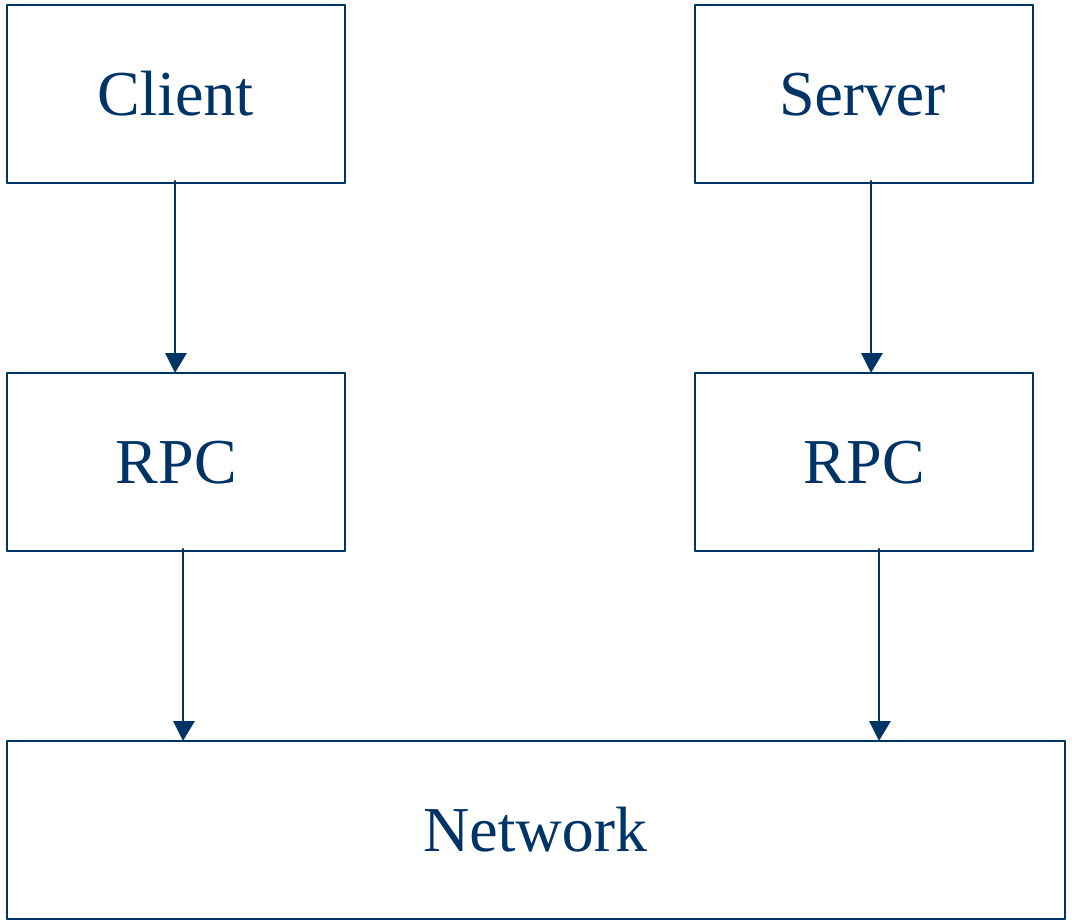
Standard Component Tricks

- Replace components
  - Different performance/size/reliability tradeoffs
  - Adapt to different hardware
- Insert component
  - Monitoring
  - Caching
  - Optional functionality
- Rearrange components
  - Where to put the cache?

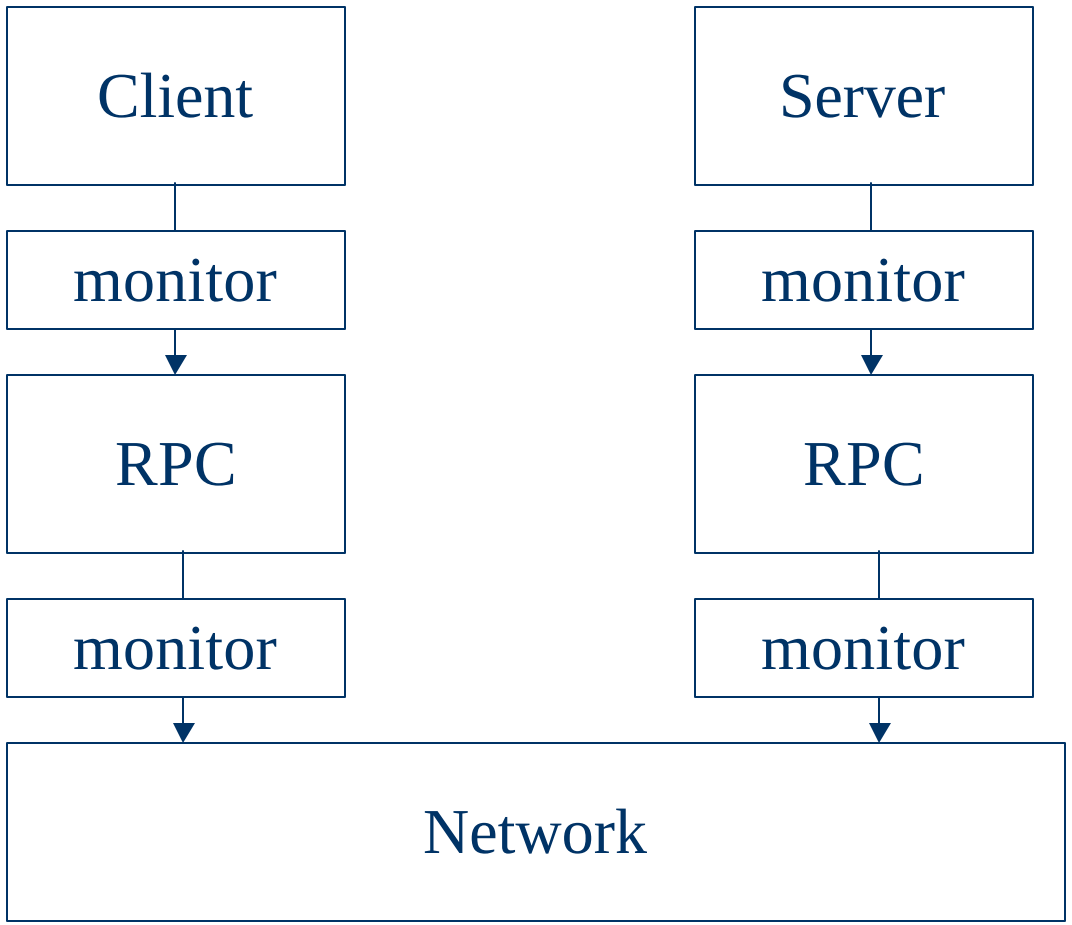
Components and aspects

- Monitoring code
- Separation/Isolation
  - Catching component failure
  - Protection zones
- Garbage collection
- Concurrency

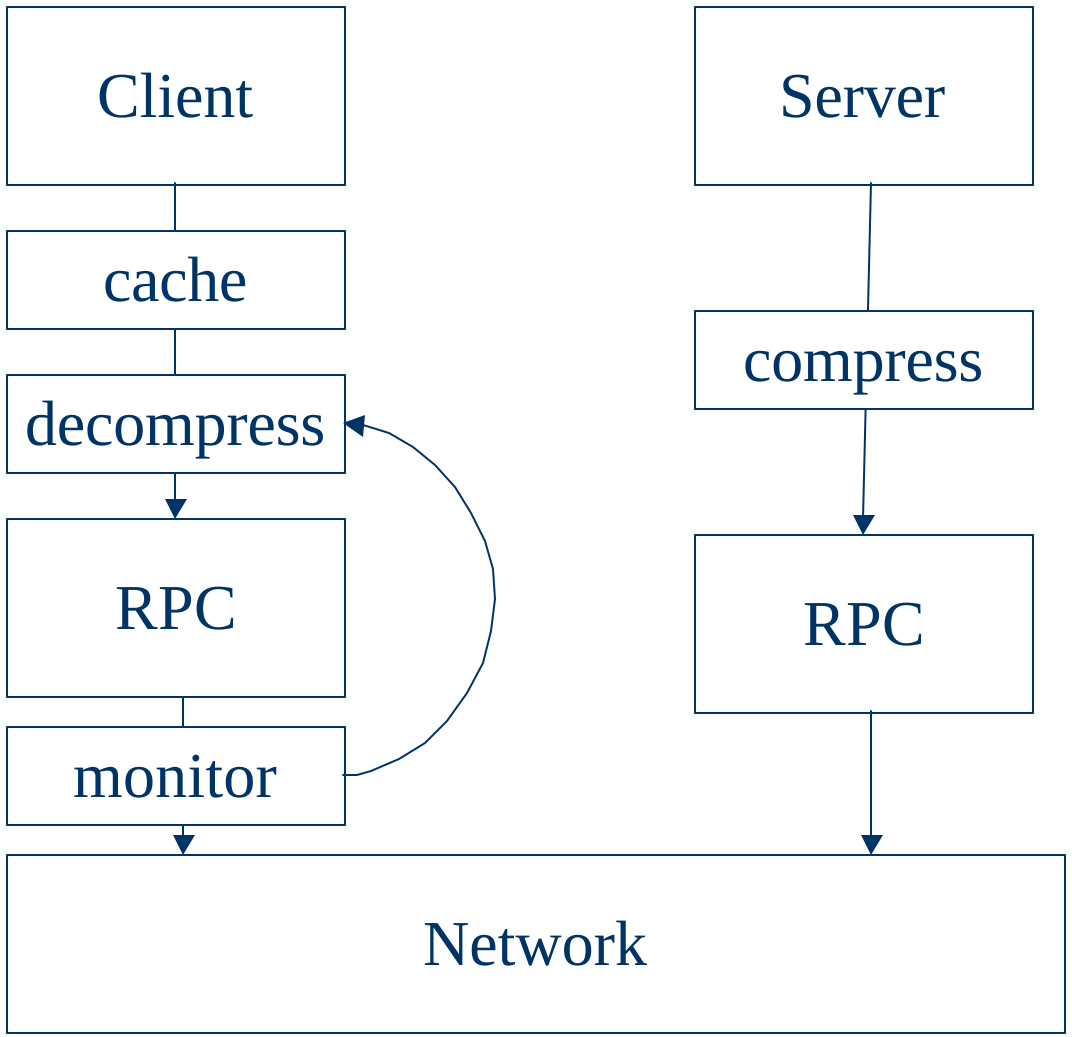
A simple system



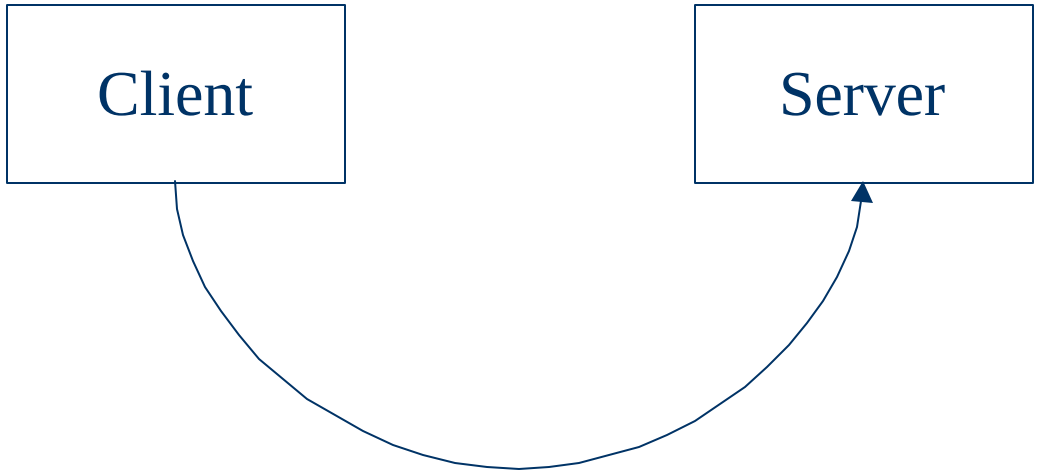
Find bottlenecks



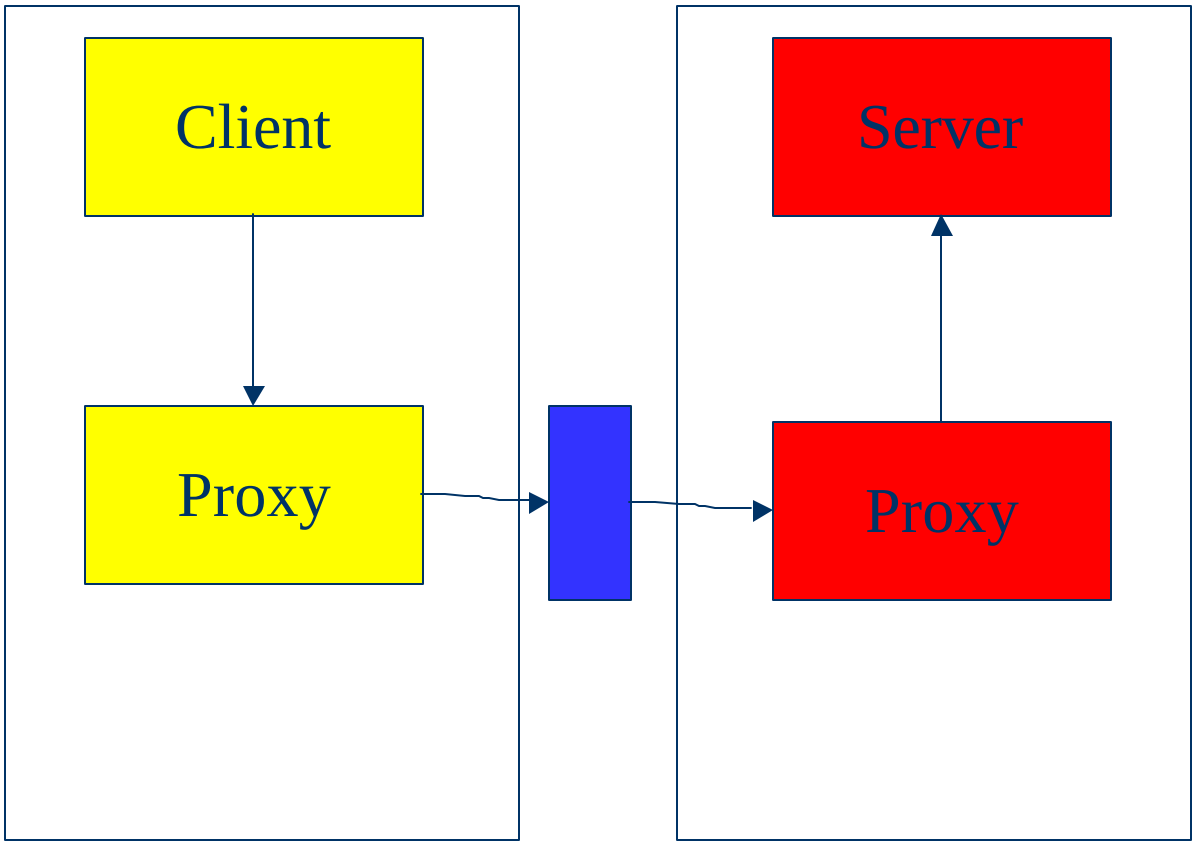
Fix the problem



Co-locate components



… but maintain some protection



Barriers to Component Programming

- Paucity of true component languages
- Cost of switching to new language
- Checking Component Configurations
- Performance
- Initializing Components
  - Not a major issue in normal applications
  - Really tricky in embedded systems

Goal of Alchemy Project

To make components practical

Utah Component Languages

- Mr. Ed - Units for Scheme [PLDI'98]
- Jiazzi - Units for Java [submitted]
- Knit – Units for C [OSDI'00]

Why C?

- Small language
- Still very popular

  Number of projects in sourceforge.org by language:

  - C: 3275 projects
  - C++: 2608 projects
  - Java: 1589 projects
- Used in interesting/real/useful code:
  - Embedded systems, Linux, FreeBSD, …
  - KaffeVM (an open source JVM)
  - …

Knit: Units for C

- Works with unmodified/lightly modified C
  - Embedded system component kit - 250 components
  - KaffeVM (an opensource JVM)
- Works with new C code
  - Clack (a re-implementation of MIT's Click modular network router) – 50 components
  - Decompose complex memory allocator into many thin layers – 7 components
- Cyclic component dependencies ok
- Automatically generates initialization code
- Extensible constraint system detects configuration errors
- Cross module inlining makes small components affordable

First Public Knit Release: 14th Feb 2001

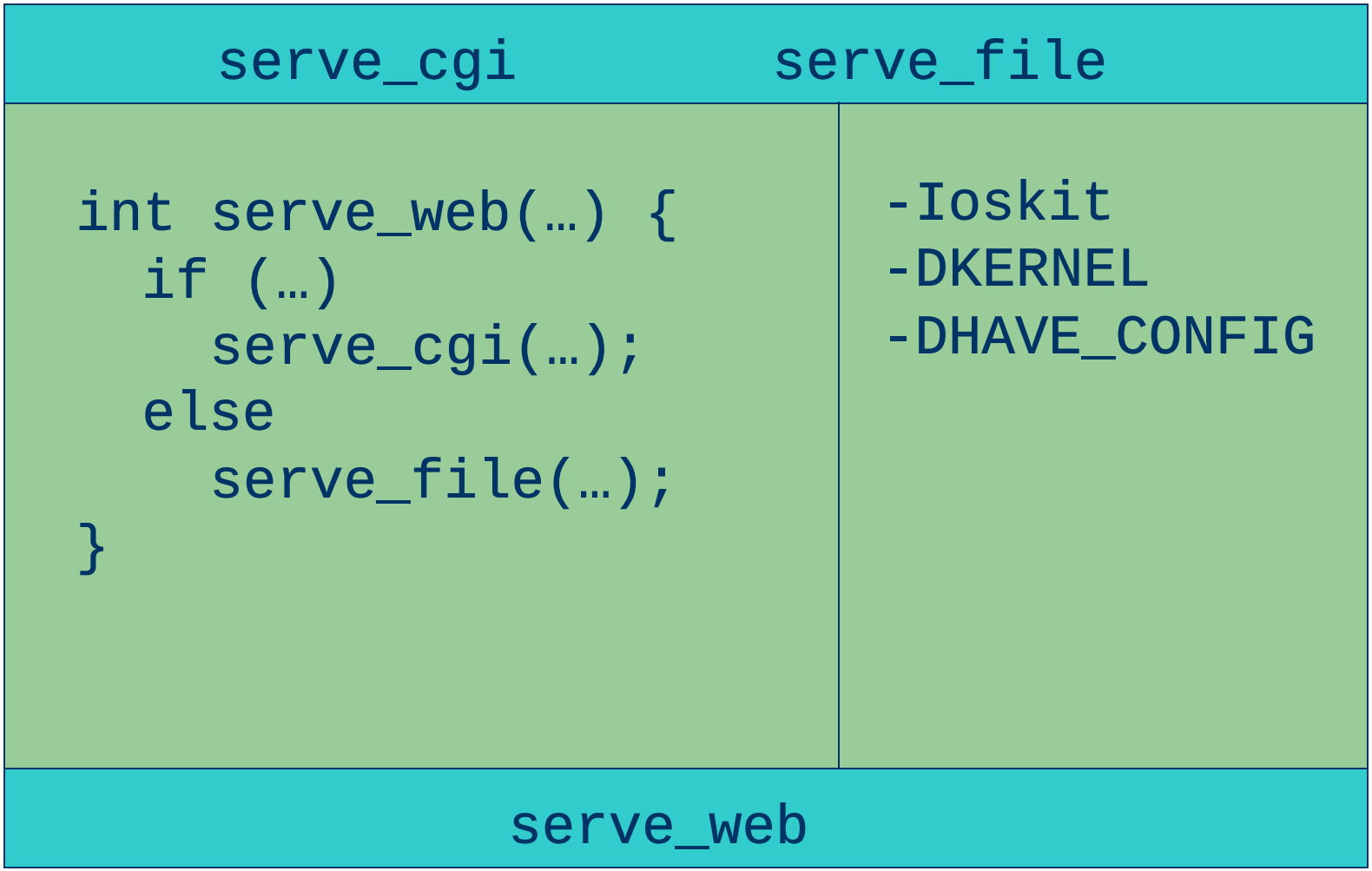
- Knit compiler
- Unit-generating tools
- Documentation generating tools
- Documentation
  - Language report
  - Tutorial
- 300 example units
- BSD-style open-source license

[See Alastair for demo today/tomorrow]

Outline

- Introduction
- Why Components?
- Components and AOP
- Barriers to Component Programming
- Knit
  - Atomic units
  - Compound units
  - Detecting Configuration Errors
  - Automatic Initialization
  - Implementation and Performance
- Status and Future Work

Atomic Units [PLDI'98]

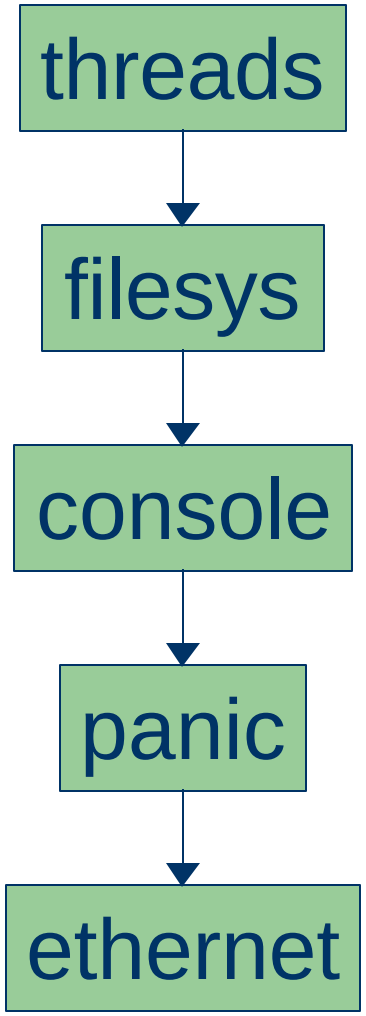


Concrete Syntax

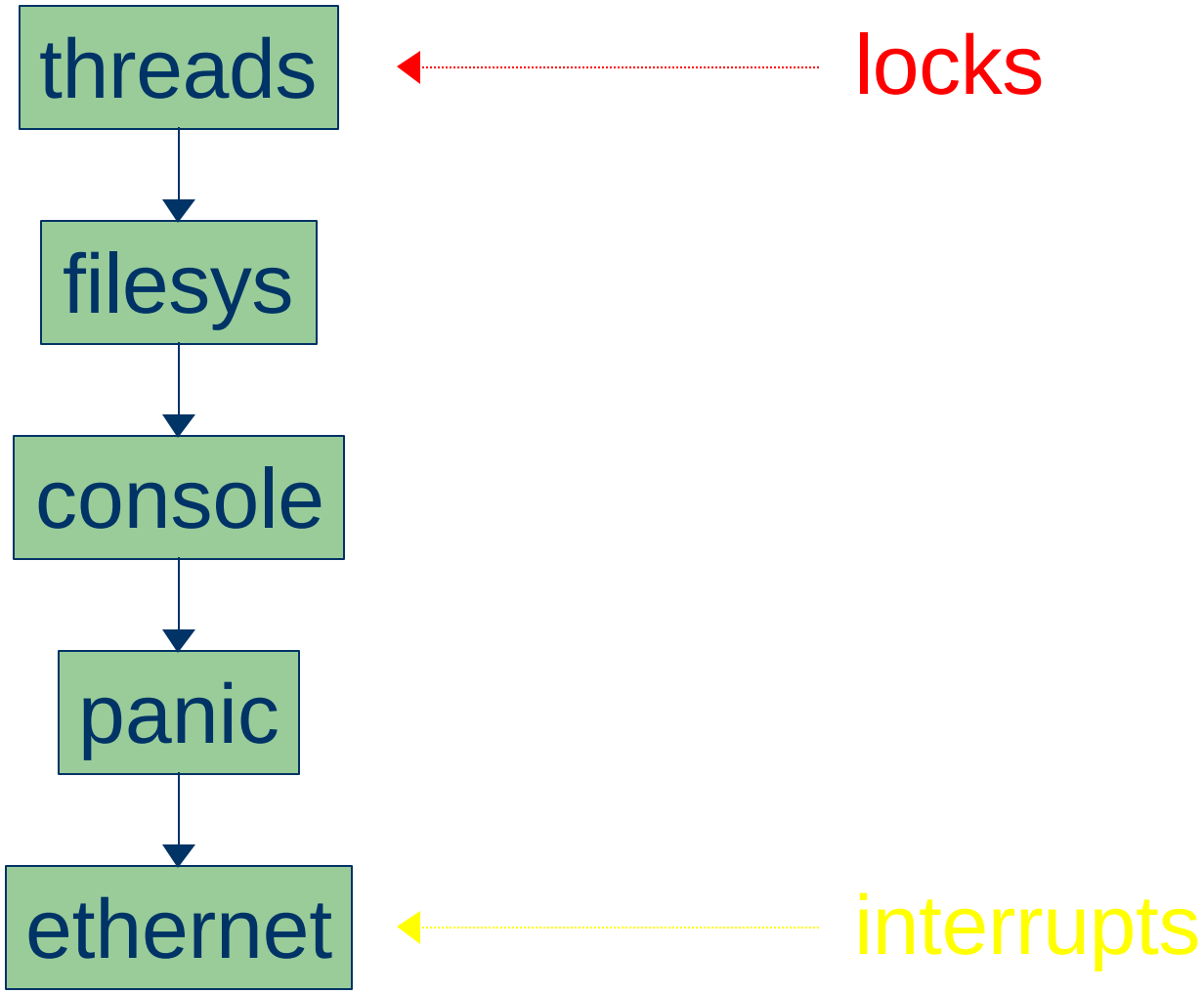
```
bundletype StdIO = { printf, ... }
bundletype Exit  = { exit, atexit }
bundletype Main  = { main }

unit hello = {
  imports[ stdio : StdIO,
           exit  : Exit ];
  exports[ main  : Main ];
  depends{ main needs imports };
  files{ "hello.c“ }
  with flags { "-Ioskit" };
}
```

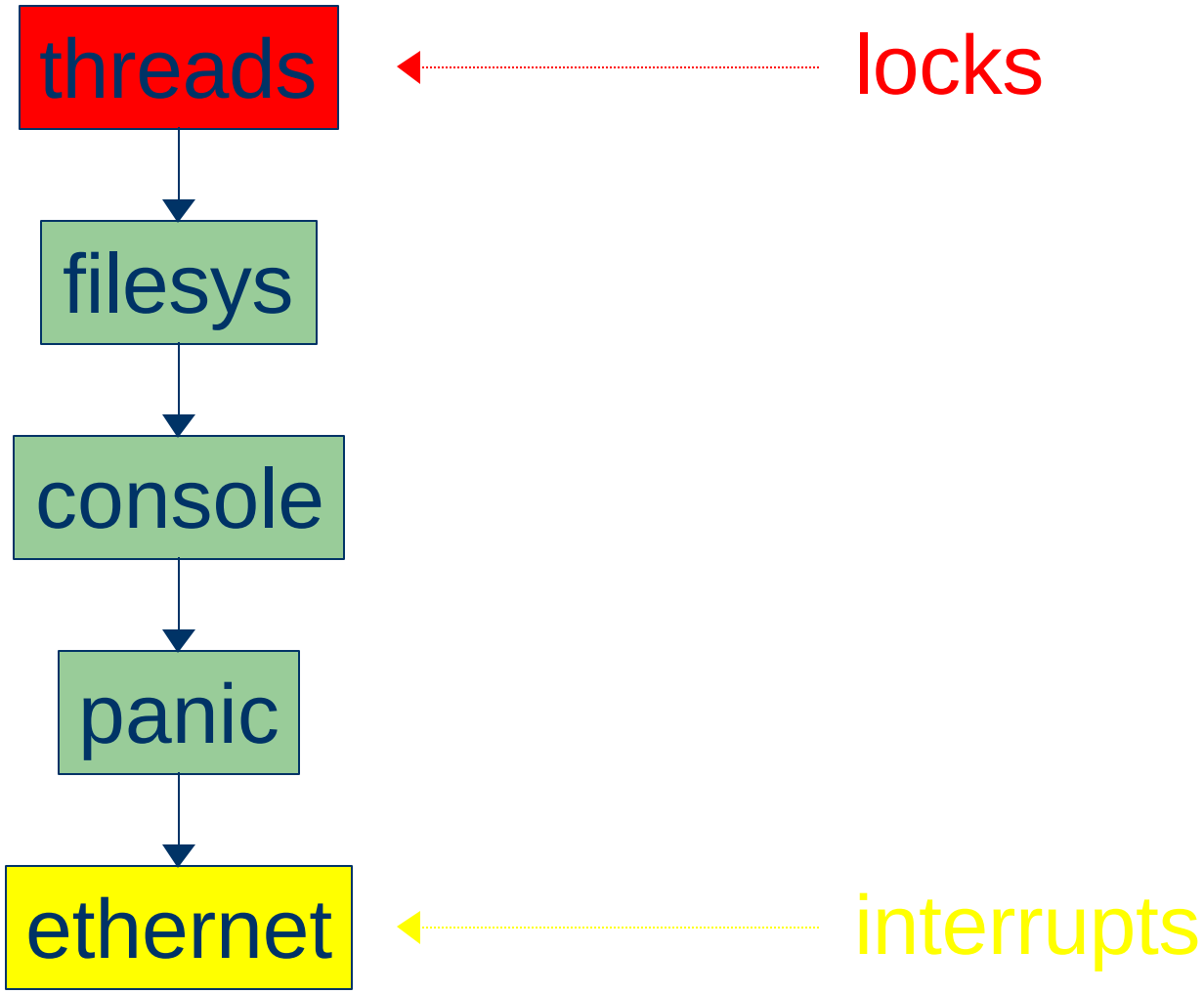
Detecting Composition Errors



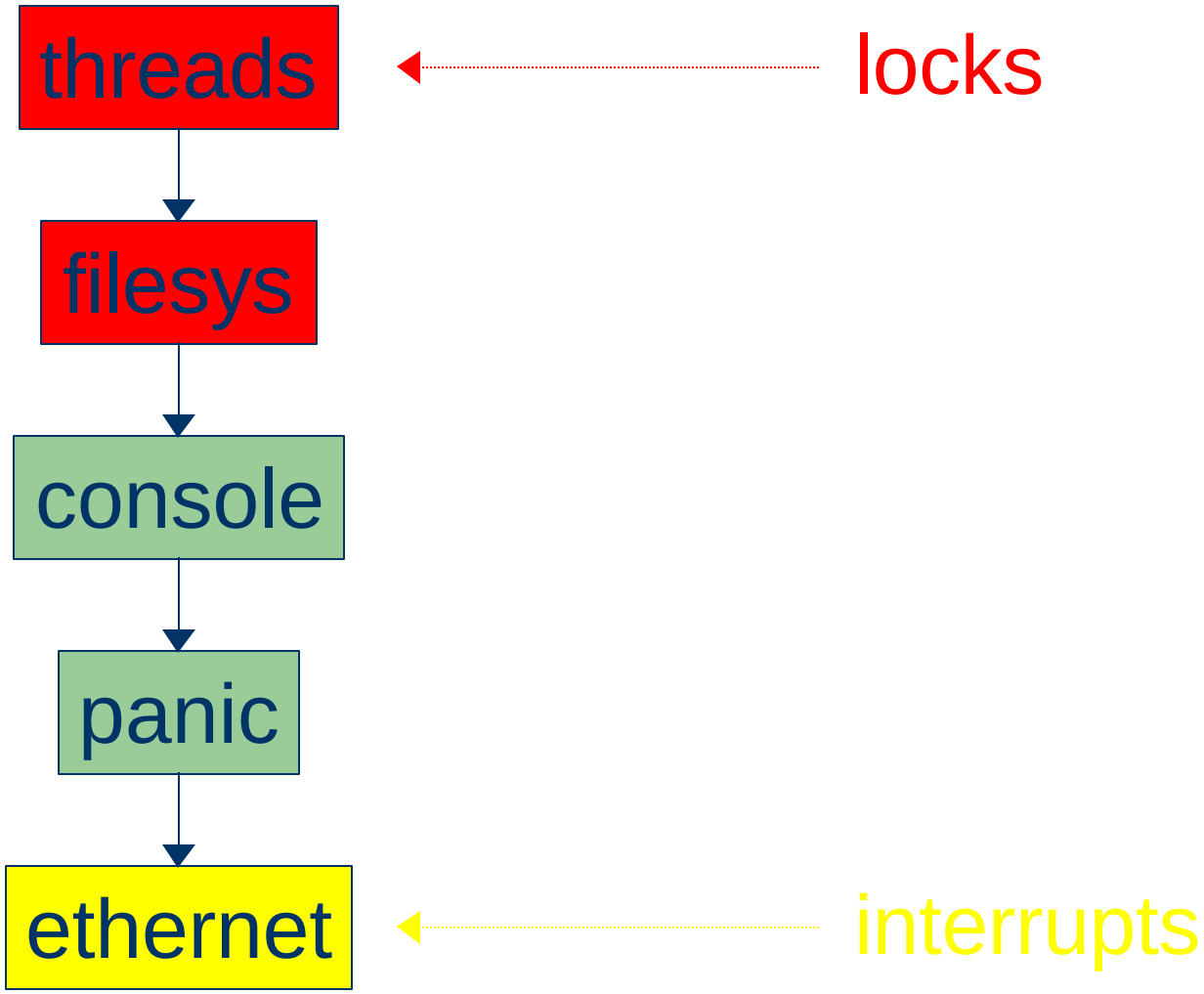
Detecting Composition Errors



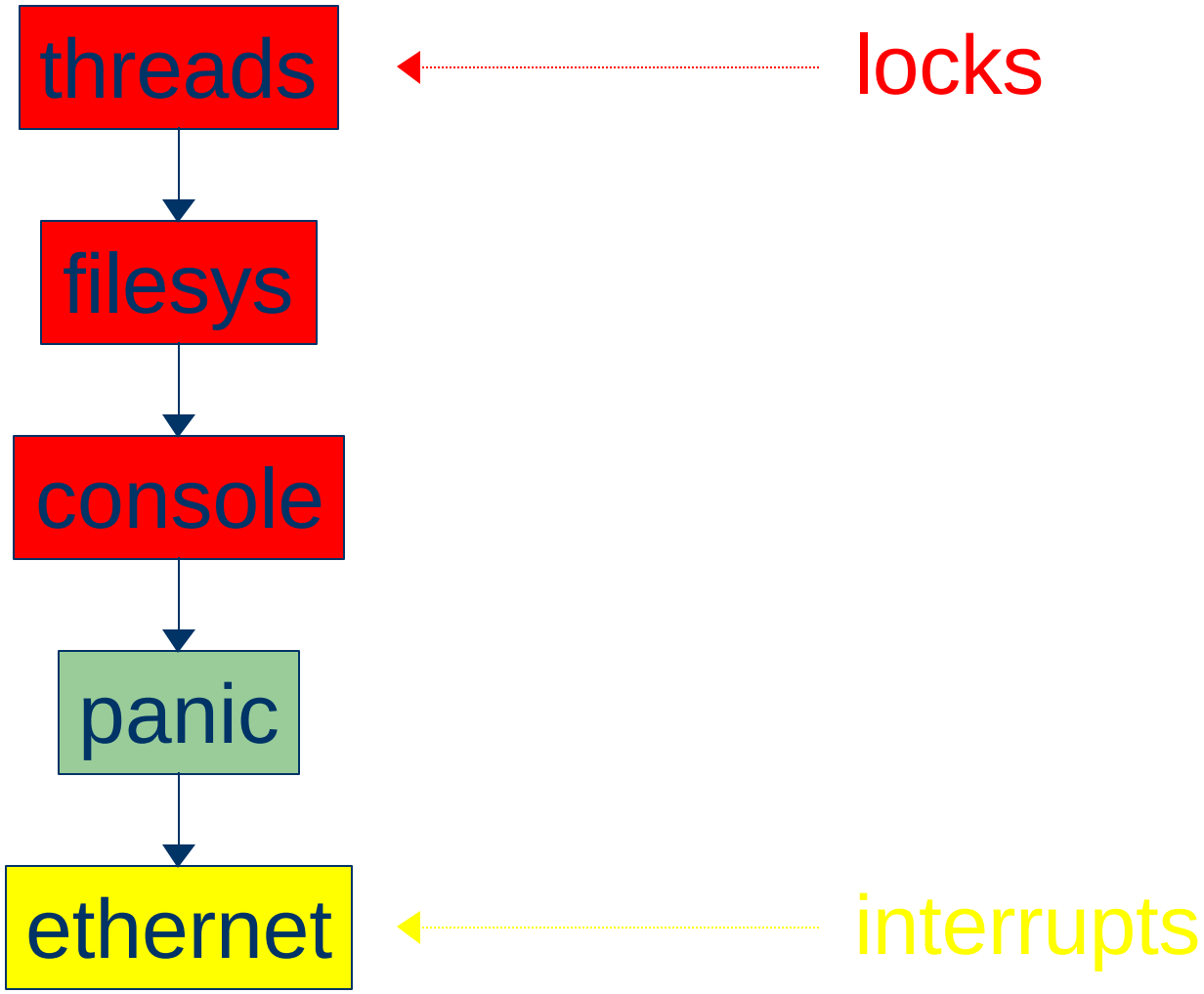
Detecting Composition Errors



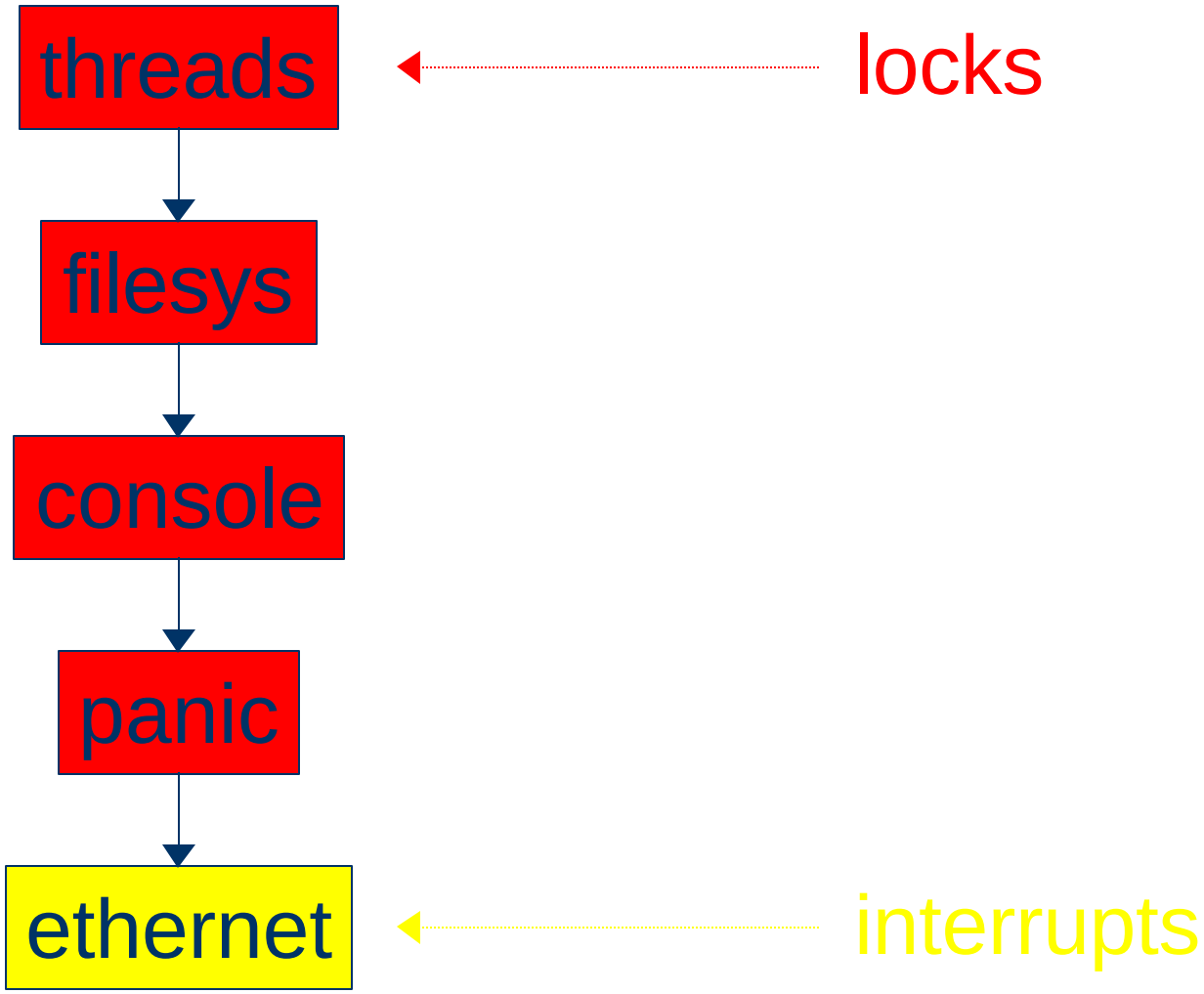
Detecting Composition Errors



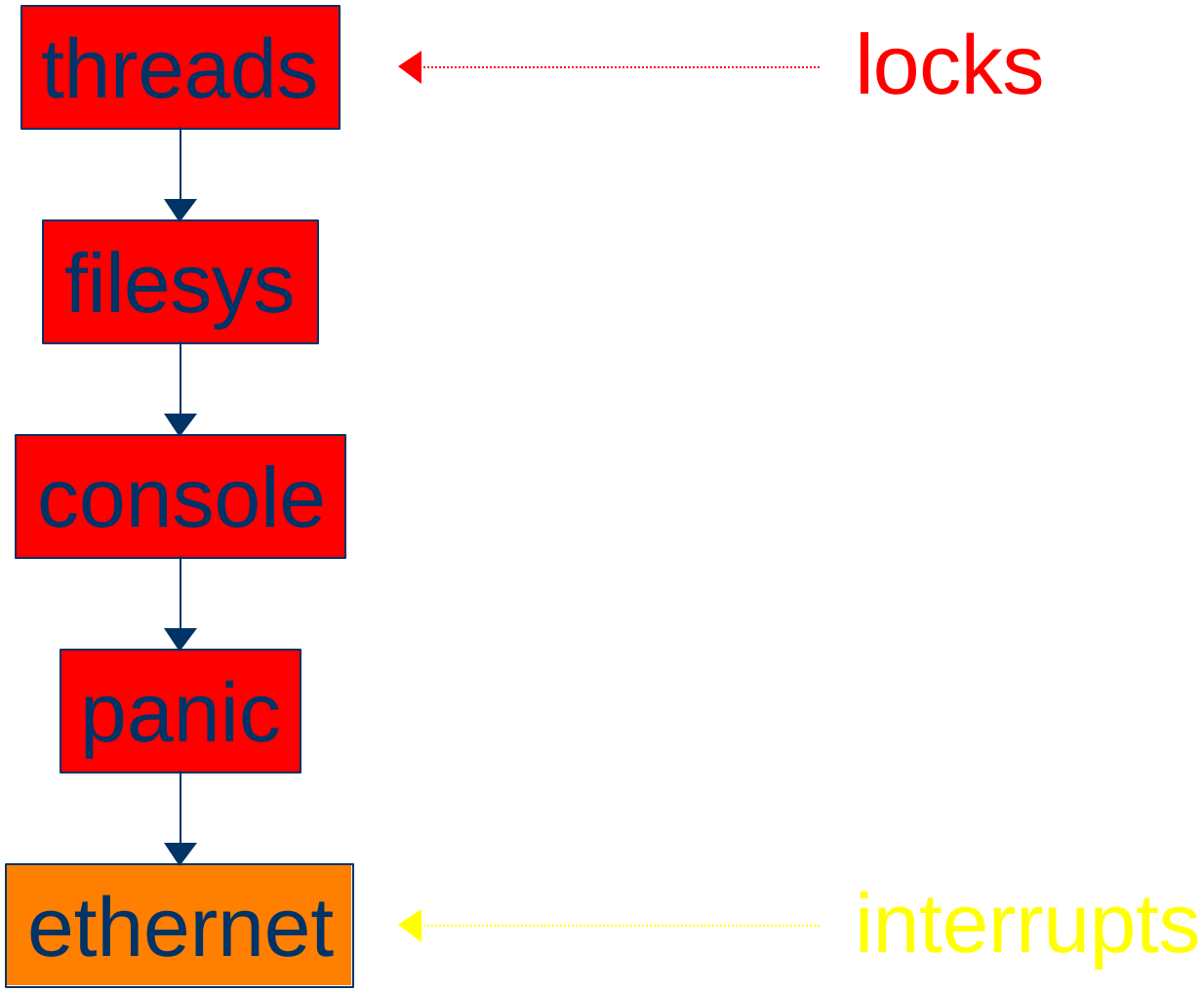
Detecting Composition Errors



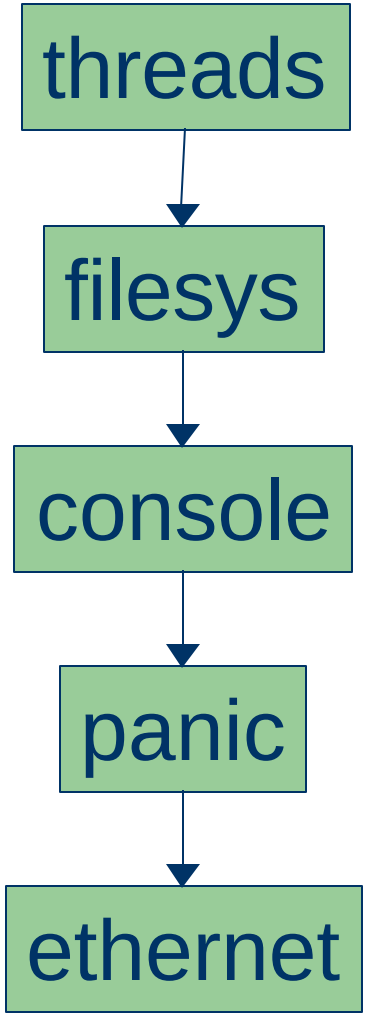
Detecting Composition Errors



Detecting Composition Errors



Detecting Composition Errors



context(threads) <= ProcessContext

context(filesys) <= context(threads)

context(console) <= context(filesys)

context(panic) <= context(console)

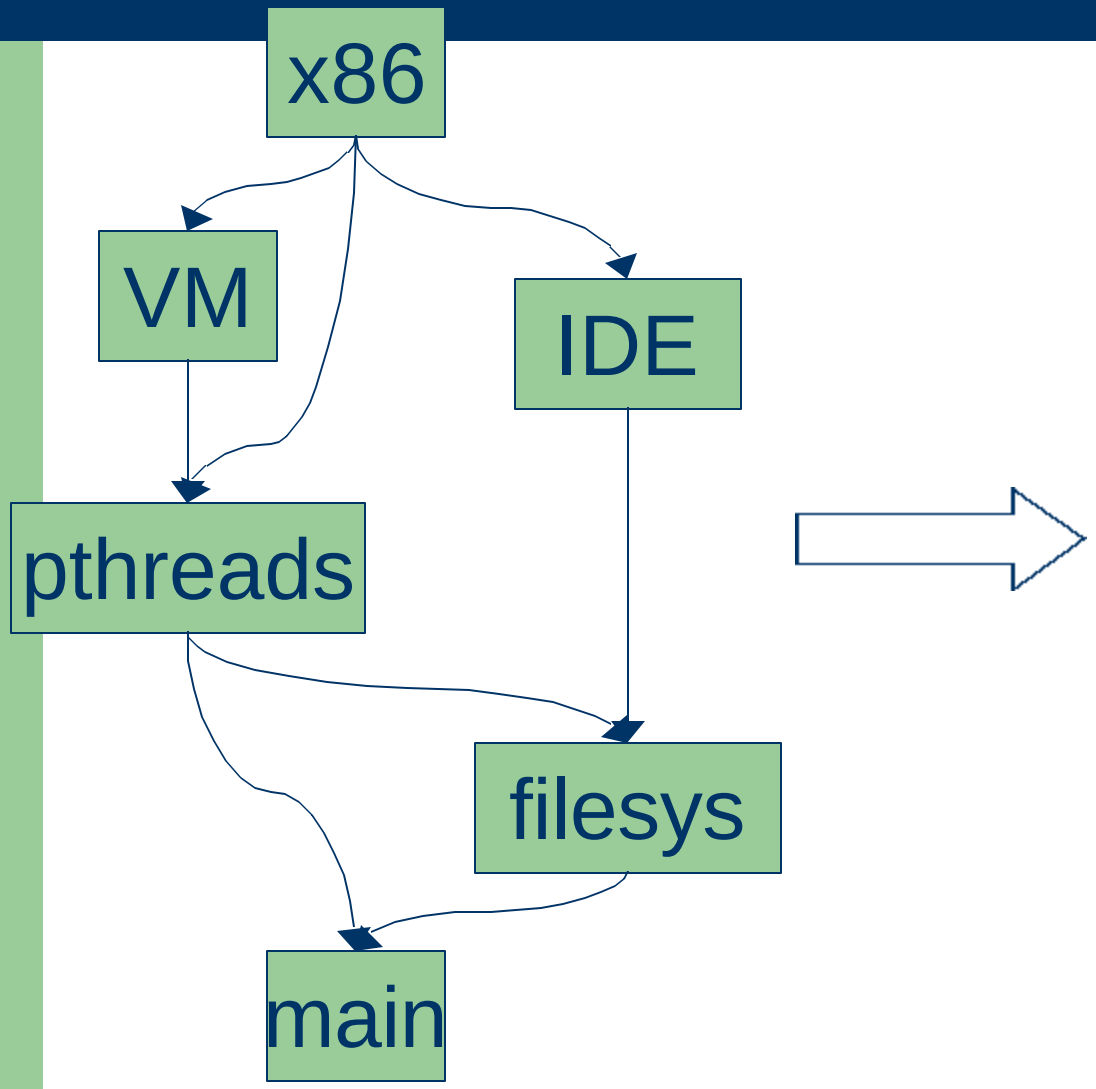
NoContext <= context(ethernet)

ProcessContext < NoContext

Extensible Constraint System

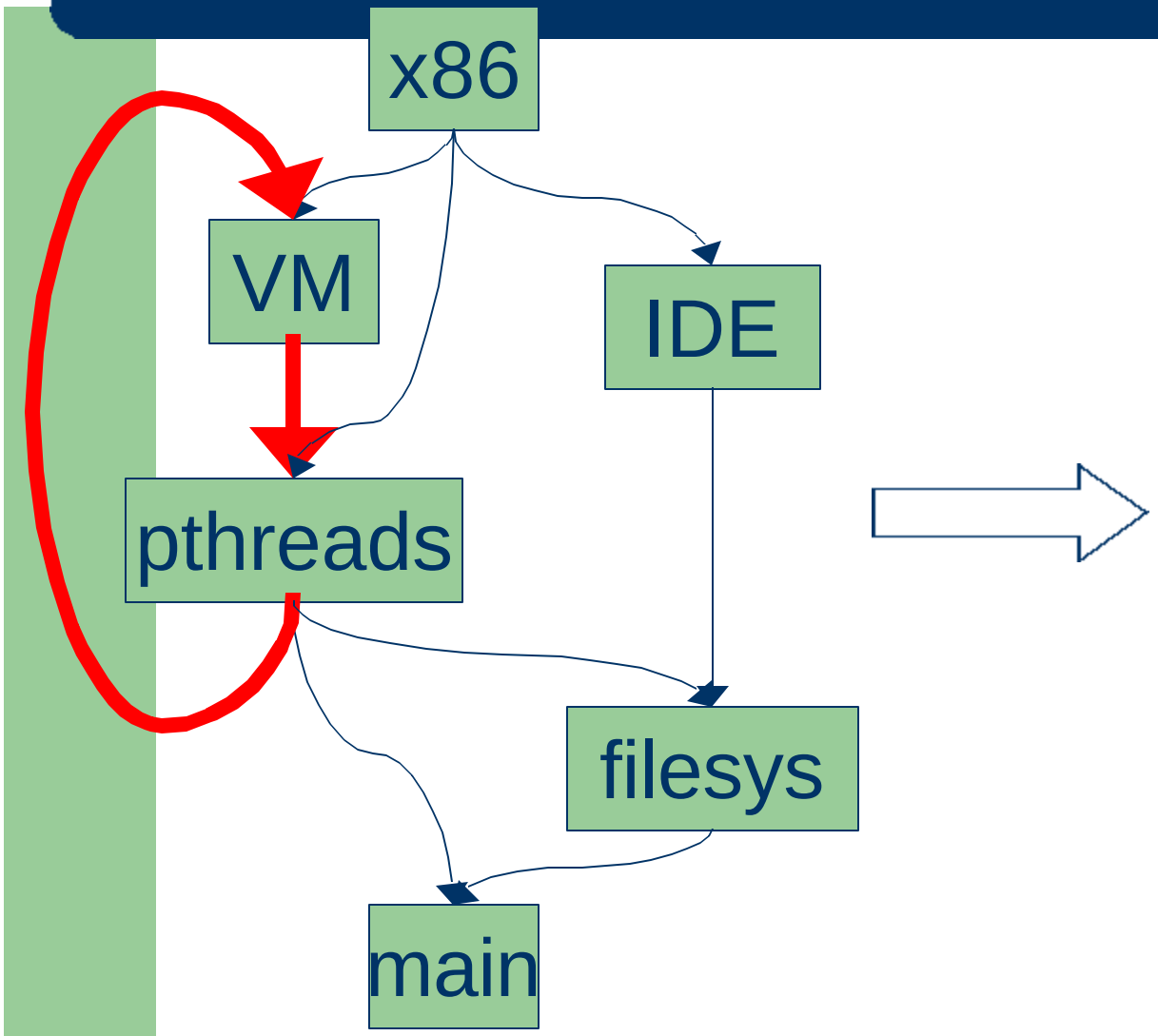
- Constraint system propagates properties through component interconnections
  - Knit can detect global errors
- Constraint system is extensible
  - In context X, don't do Y
  - Type system for Modular IP Routers (e.g., Click)
  - …

Initialization




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

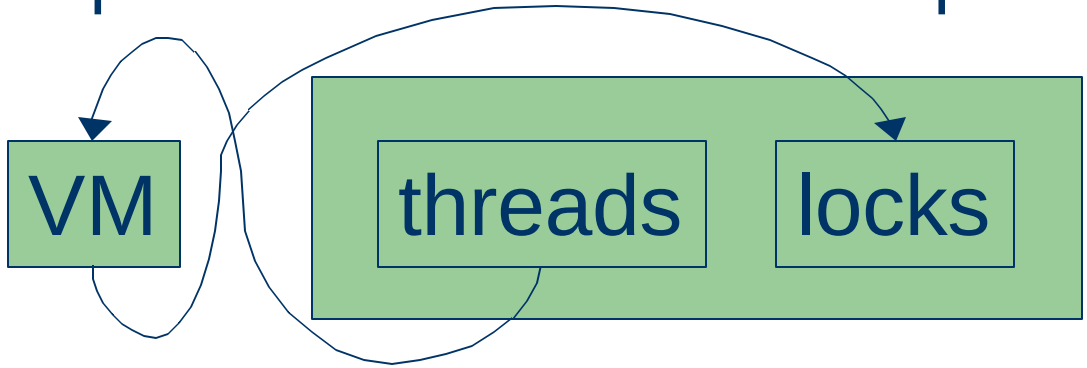
Initialization



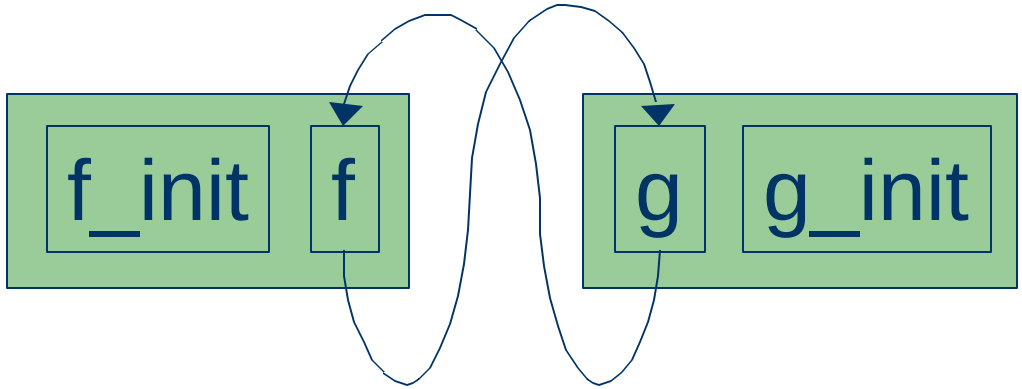

```
init_x86();
init_IDE();
init_VM();
init_threads();
init_filesys();
init_main();
```

When Can We Break Cycles?

1. Component ‘contains’ subcomponents



2. No dependency between initializers

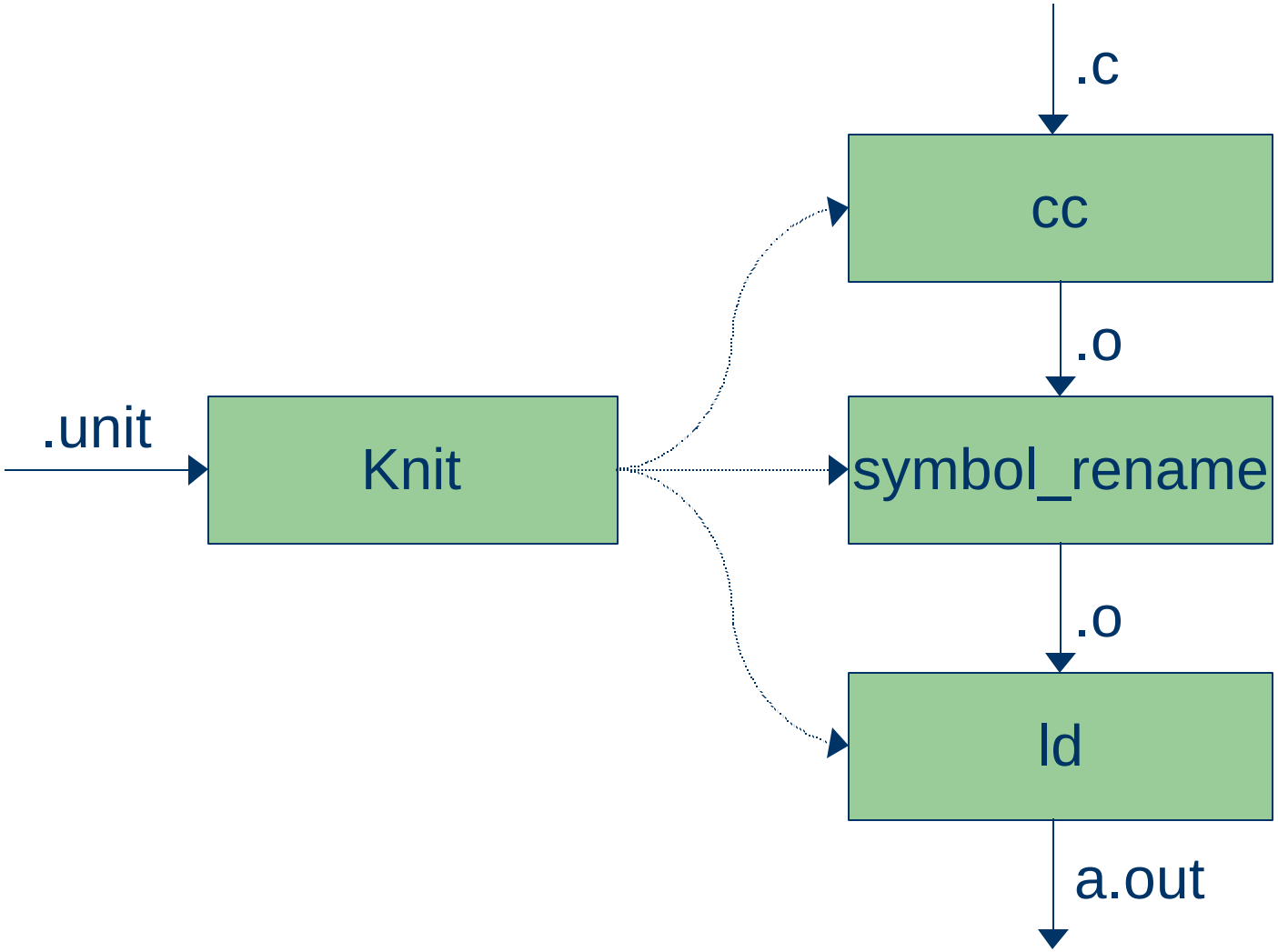


Automatic Initialization

- 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

Implementation (Unoptimized)



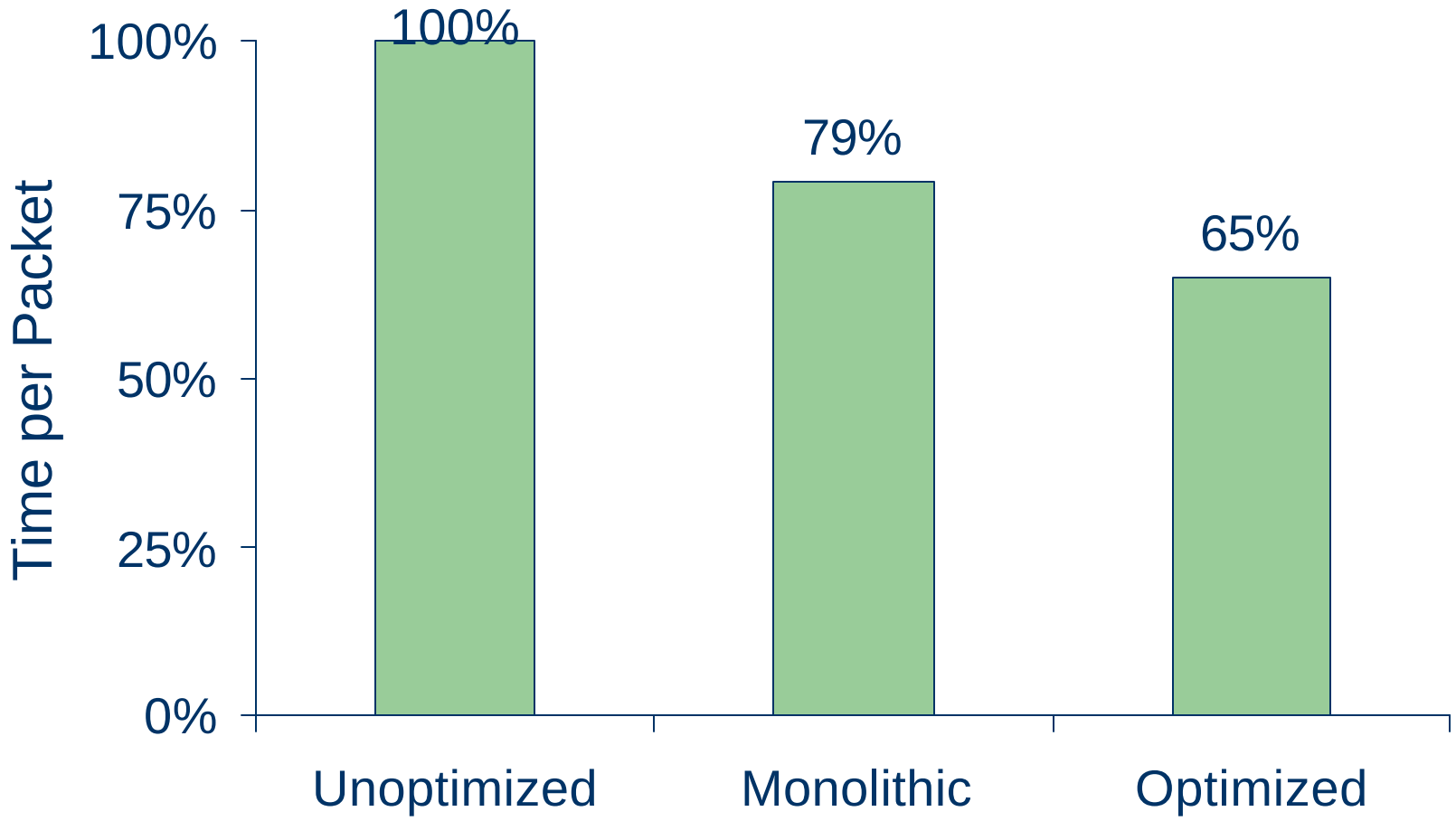
Performance

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

Click and Clack

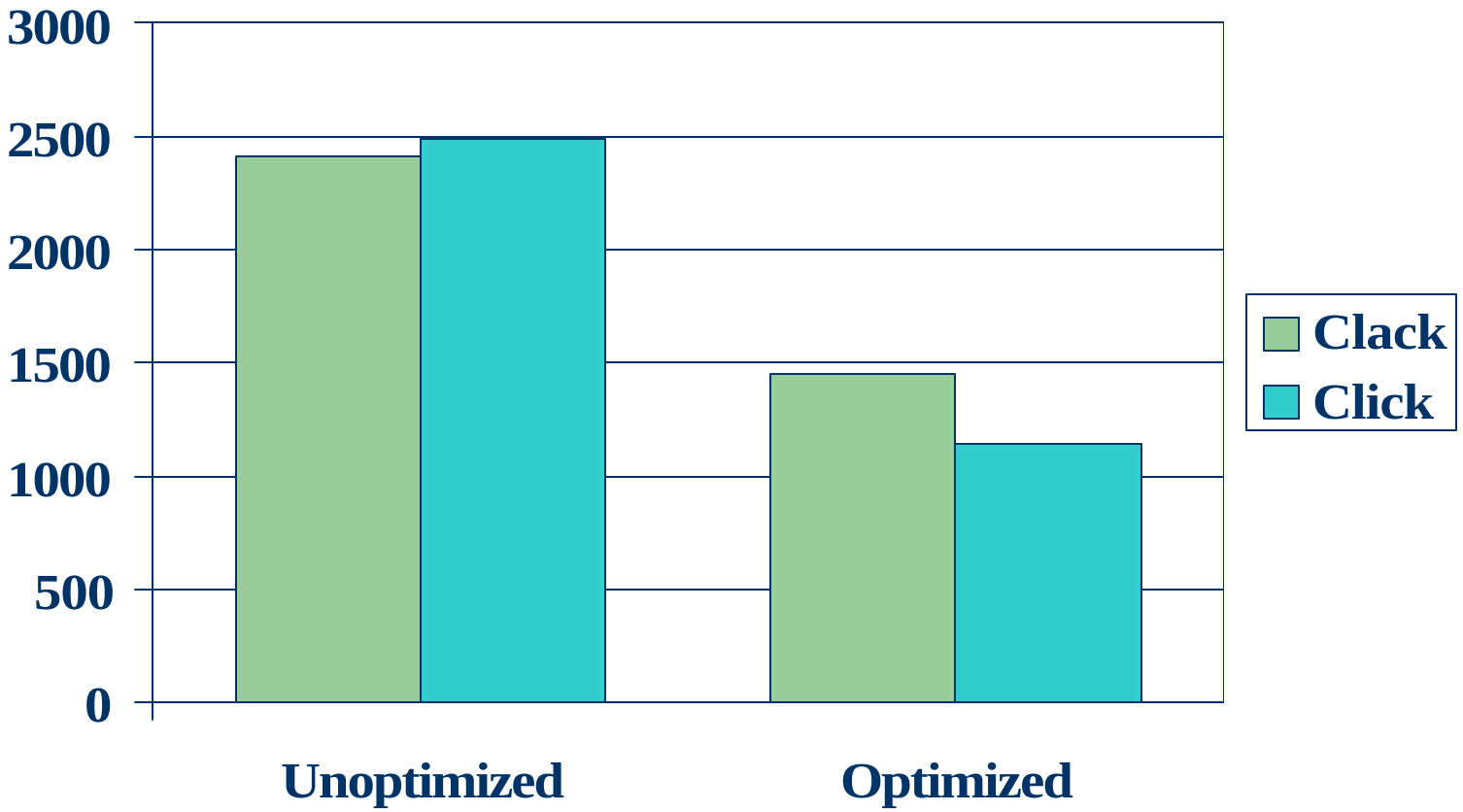
- Click modular network router from MIT [SOSP'99]
- Clack
  - Re-implementation of Click using Knit
  - Similar performance to Click
- Many small components

Performance of Clack



Knit

Click vs. Clack Performance



Knit

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

Current Status

- First public Knit release next week
- 300 embedded system components
- Constraint systems
  - Top/bottom-half code
  - Types of network packets

Future Work

- Constraints
  - Real time constraints
    - Restrictions on real time threads
    - Timing
  - Scaling issues
  - Hooks for external code analyzers
  - Hooks for external constraint systems

Future Work

- Properties/Aspects
  - Isolation
    - Protection domains
    - Detect component failure
    - Recover from failure
  - Performance monitoring and adaptation
    - Monitor resource use: time, bandwidth, memory, …
    - Feedback into scheduler/application/network stack/etc.
  - Memory Management
  - Concurrency

Future Work

- Weave components through configurations
- Automatically generate components
  - proxies (caching, ...)
  - adapters (RPC, protection domains, GC, …)
  - advice (monitoring, logging, …)