Alchemy:
Transmuting Raw Code into Reusable Components

Jay Lepreau (PI)
Matthew Flatt (co-PI)
Wilson Hsieh
Jeanette Wing CMU

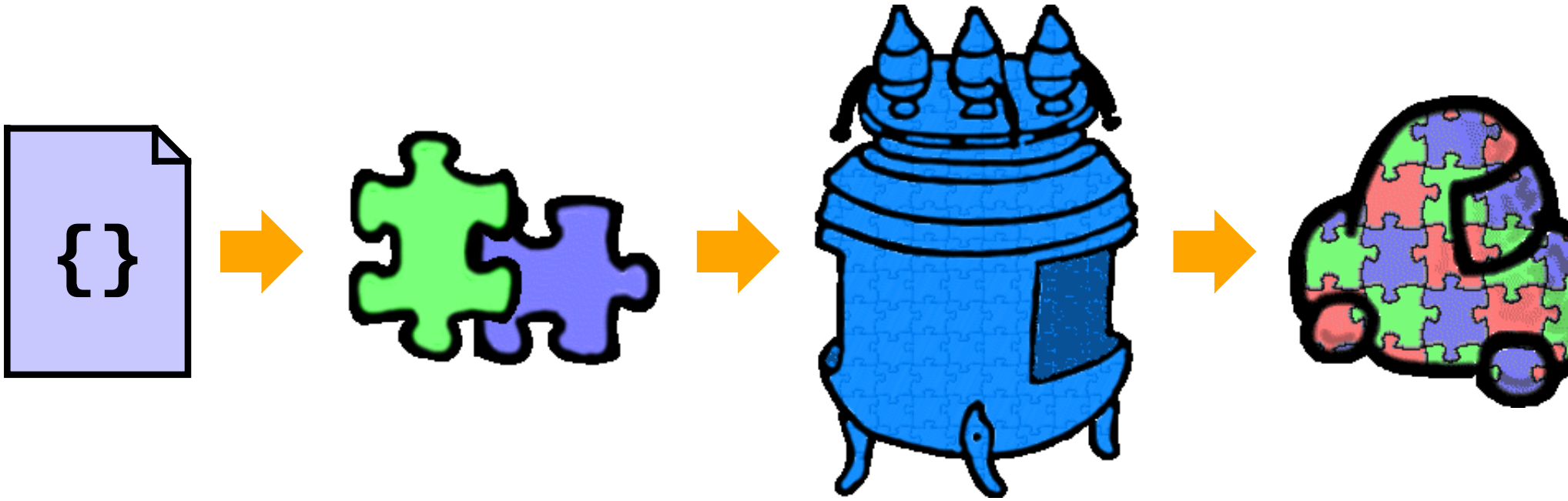
Alastair Reid
Leigh Stoller
Eric Eide
Mike Hibler

University of Utah

http://www.cs.utah.edu/flux/alchemy/

Alchemy

Component language and tools
for low-level systems software

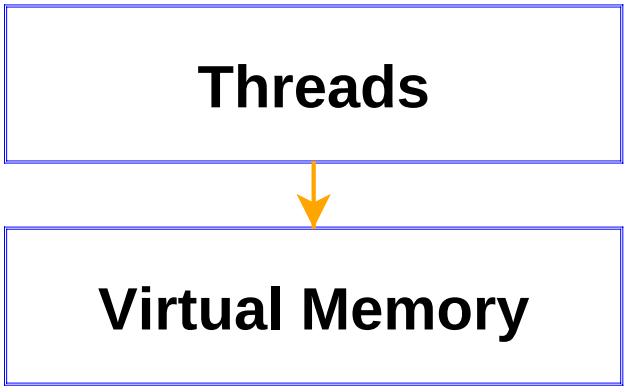


Not a specific architecture or framework

Example: **The OSKit** [SOSP97]

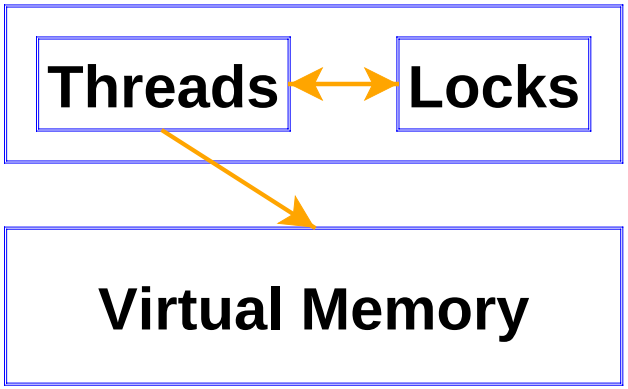
Characteristics of Low-Level Systems

- Lack of clear layers among components



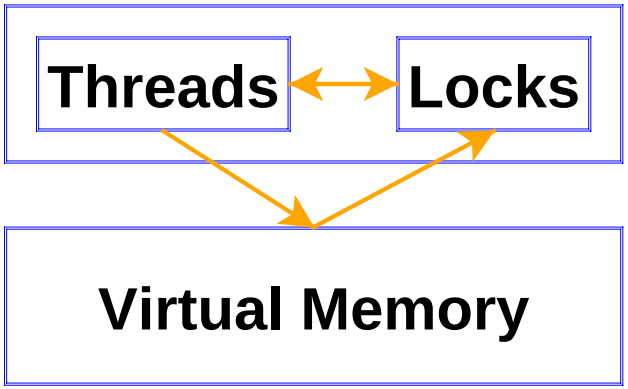
Characteristics of Low-Level Systems

- Lack of clear layers among components



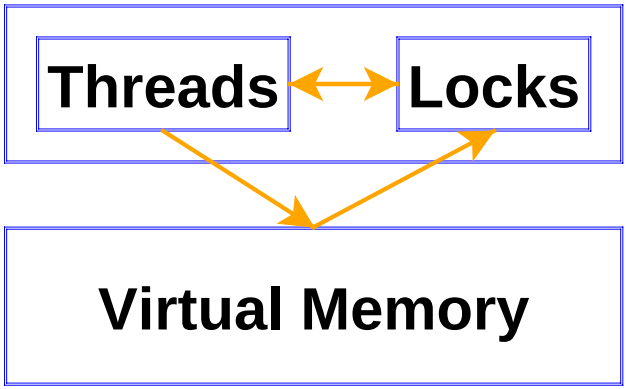
Characteristics of Low-Level Systems

- Lack of clear layers among components

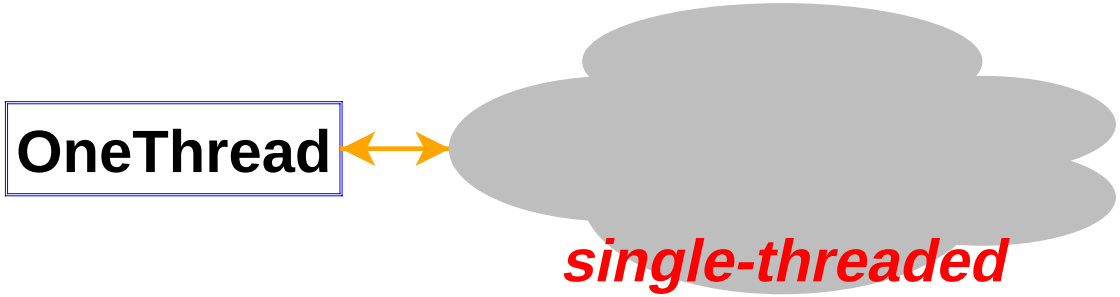


Characteristics of Low-Level Systems

- Lack of clear layers among components

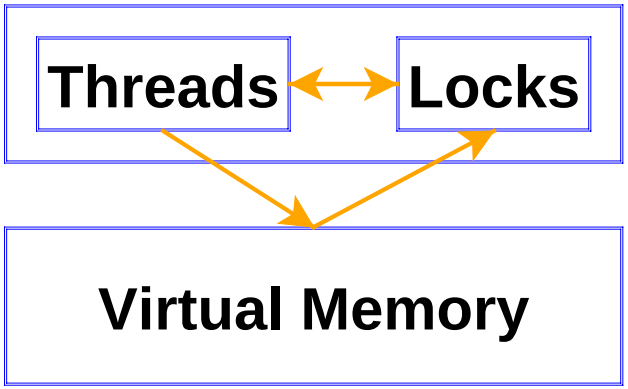


- Local changes shift global constraints

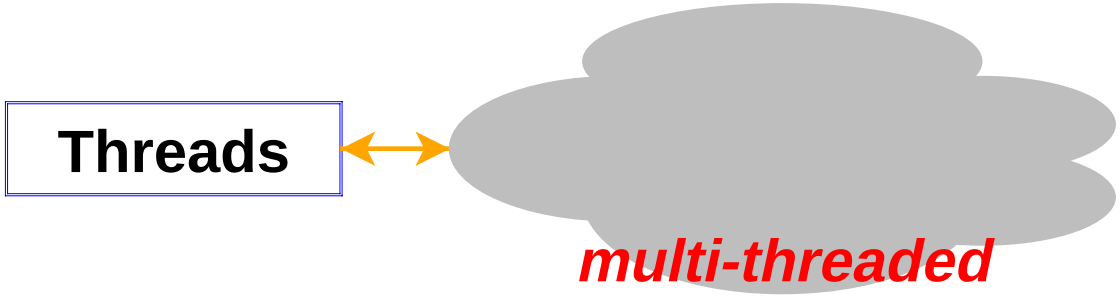


Characteristics of Low-Level Systems

- Lack of clear layers among components



- Local changes shift global constraints



Alchemy Outline

- **Language**
  - define components
  - link components
- **Types and constraints**
  - verify linking
  - infer adaptors
- **Performance**
  - zero componentization overhead
- **Applications**
  - The OSKit
  - Linux, ...

Language

Start with **unit** model of components [PLDI98]

$\mathbf{import}_1$ ... $\mathbf{import}_n$
$\mathbf{definition}_1$ ... $\mathbf{definition}_m$
$\mathbf{export}_1$ ... $\mathbf{export}_k$

atomic unit shape

Language

Start with **unit** model of components [PLDI98]

```
lock_t  void lock(lock_t)  ...
```

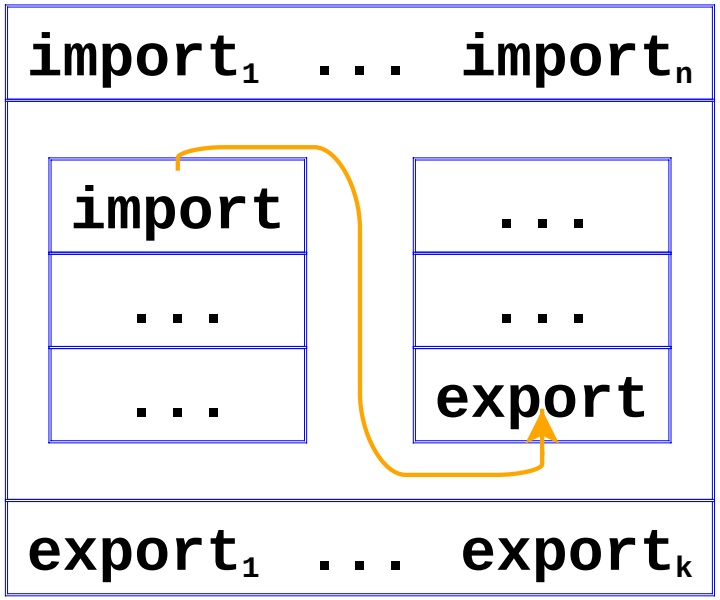
```
void* mmap(long s) { ... }
...
```

```
    void* mmap(long)  ...
```

atomic unit example

Language

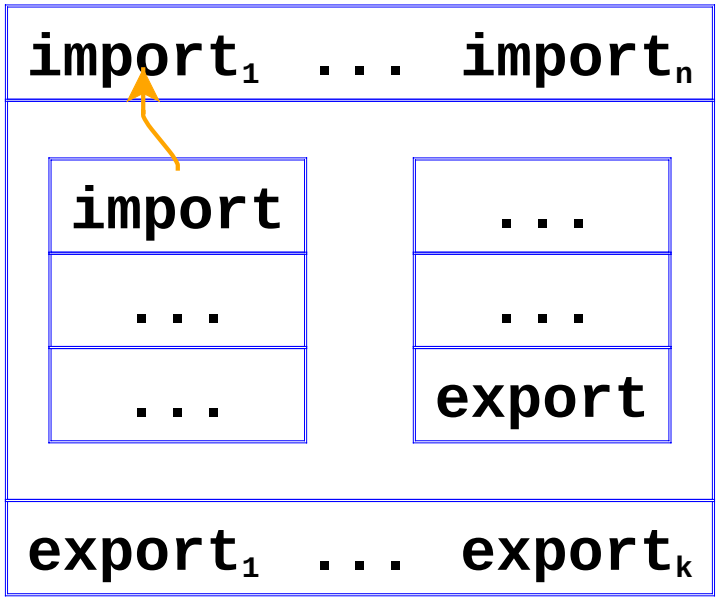
Start with **unit** model of components [PLDI98]



compound unit shape

Language

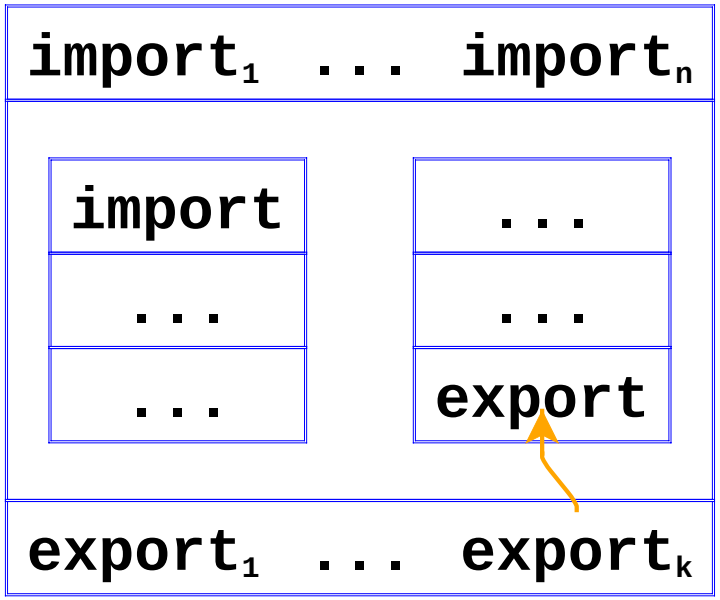
Start with **unit** model of components [PLDI98]



compound unit shape

Language

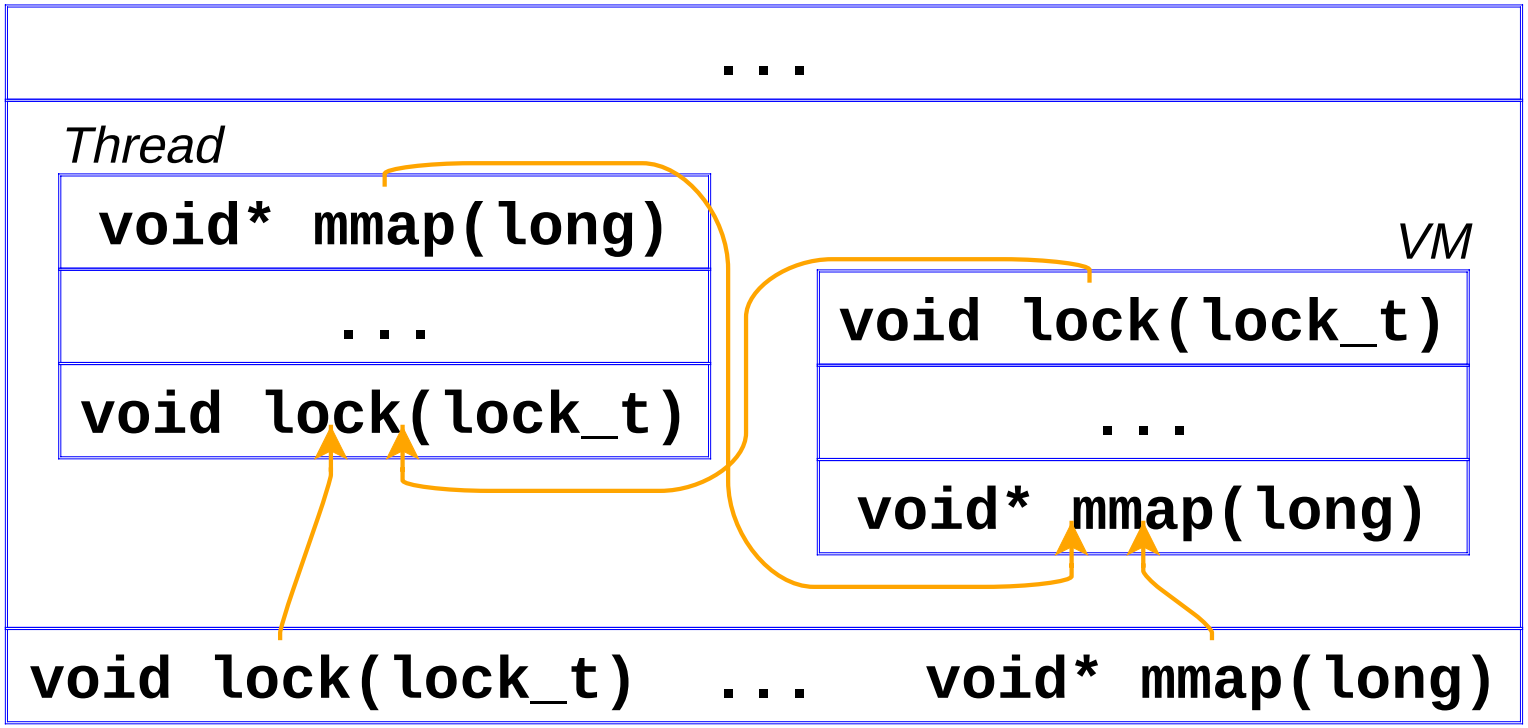
Start with **unit** model of components [PLDI98]



compound unit shape

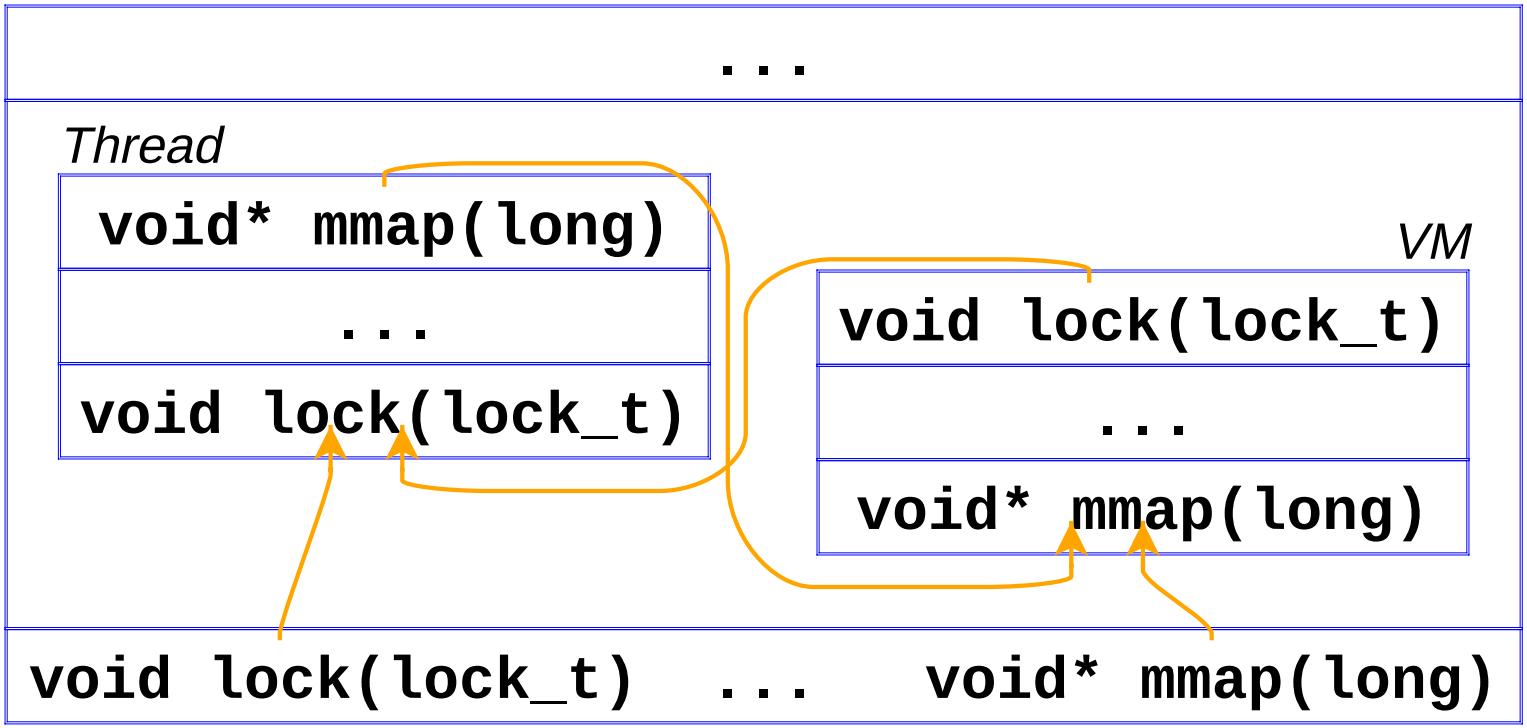
Language

Start with **unit** model of components [PLDI98]



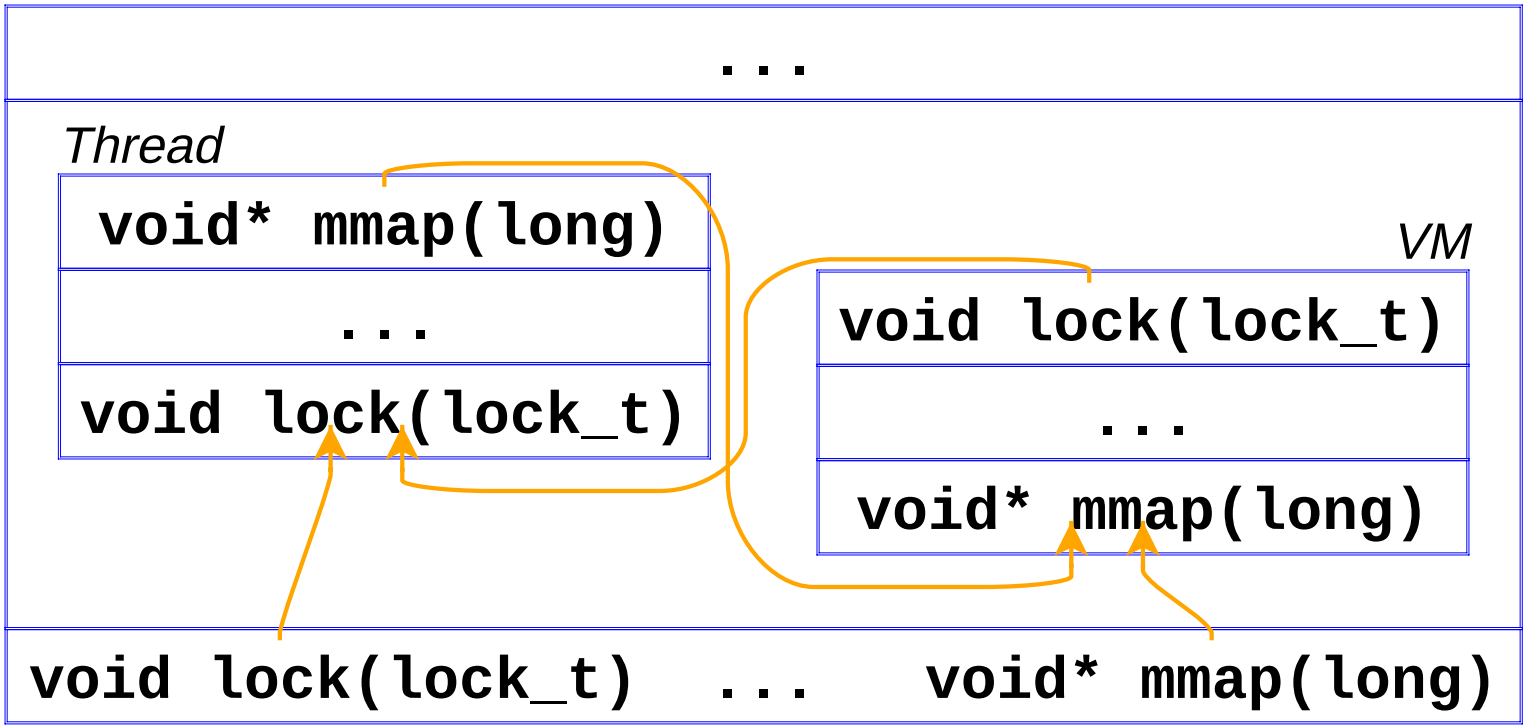
compound unit example

Language



- well-defined interfaces,
- hierarchical,
- mutual dependencies,
- multiple instances, ...

Language



... and **static**!

Usability

Unit language for C:

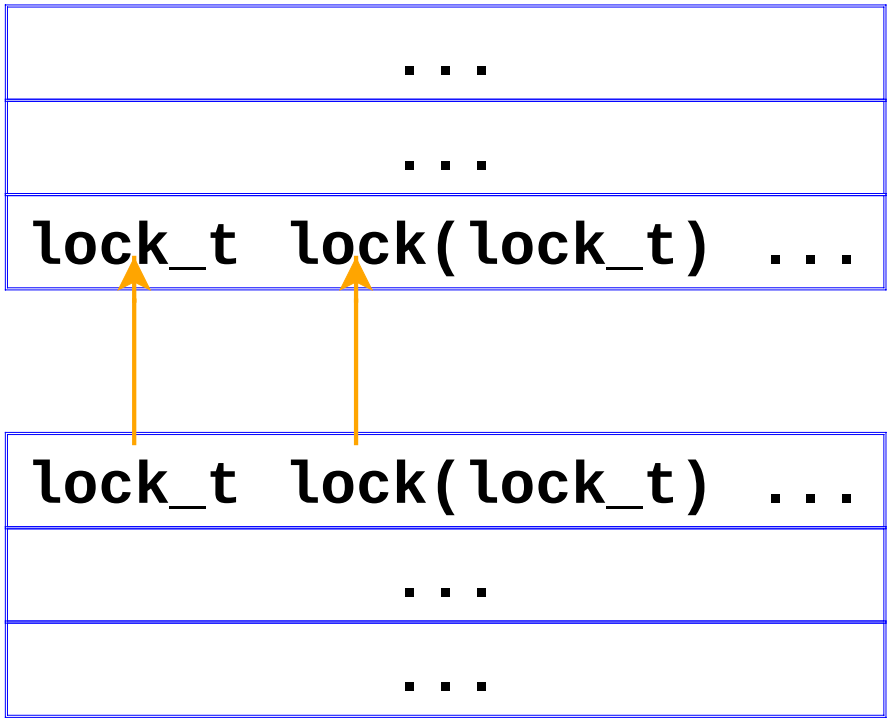
- Must be easy for system hackers to use
- Must provide a mechanism for deriving configurations

Alchemy Outline

- **Language**
  - define components
  - link components
- **Types and constraints**
  - verify linking
  - infer adaptors
- **Performance**
  - zero componentization overhead
- **Applications**
  - The OSKit
  - Linux, ...

Types and Constraints

Unit model provides basic type checking:



Types and Constraints

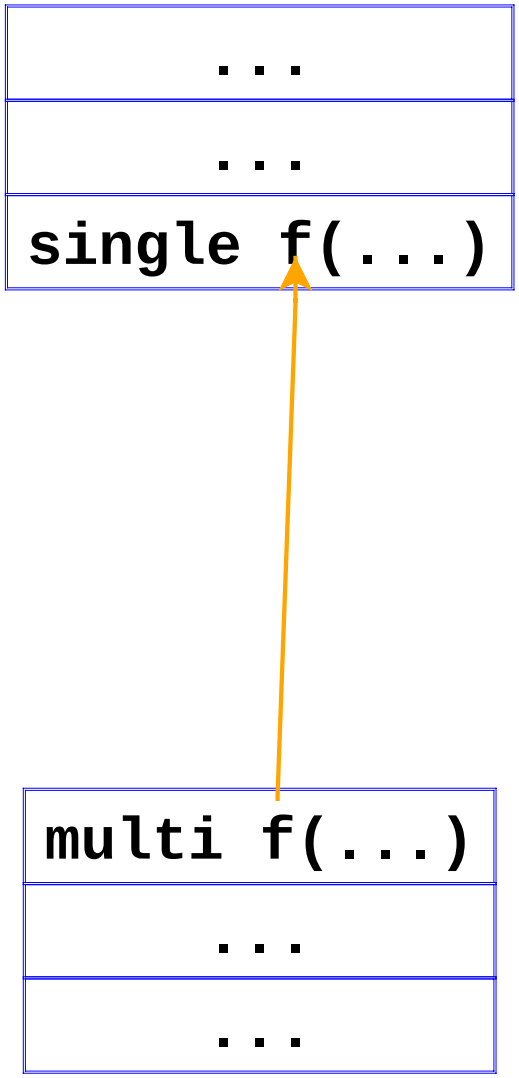
For systems software, also need to ensure that

- Multi-threaded code doesn't call single-threaded code
- Only one virtual memory manager is active
- Components are initialized in the correct order
- Components obey resource constraints
- *and more*

These are extra constraints, outside "normal" types

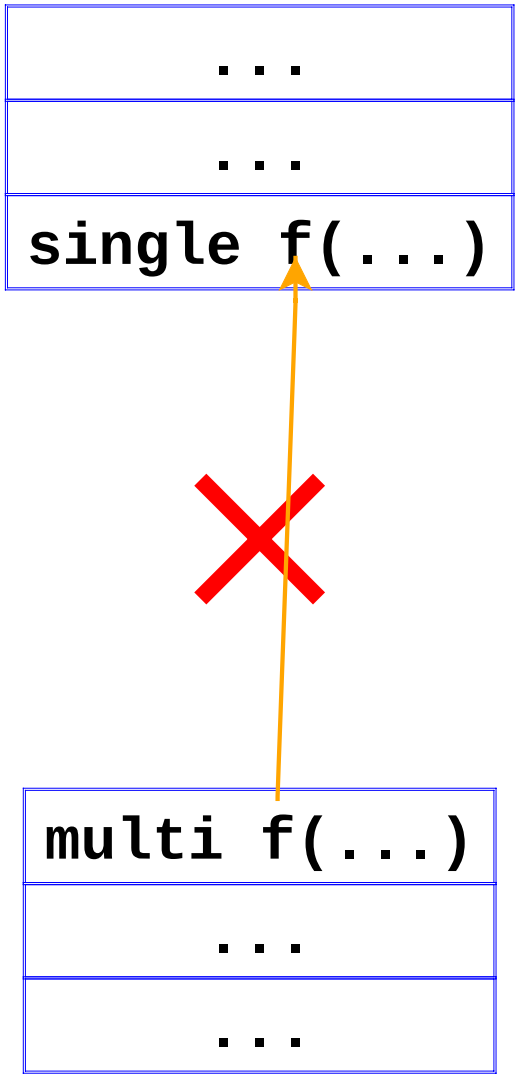
Constraint Checking

Reject mismatches:



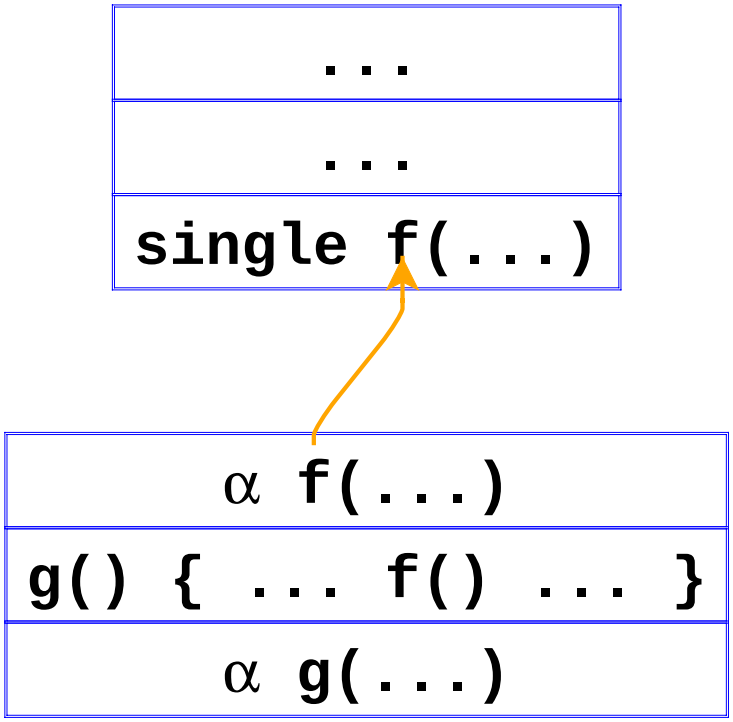
Constraint Checking

Reject mismatches:



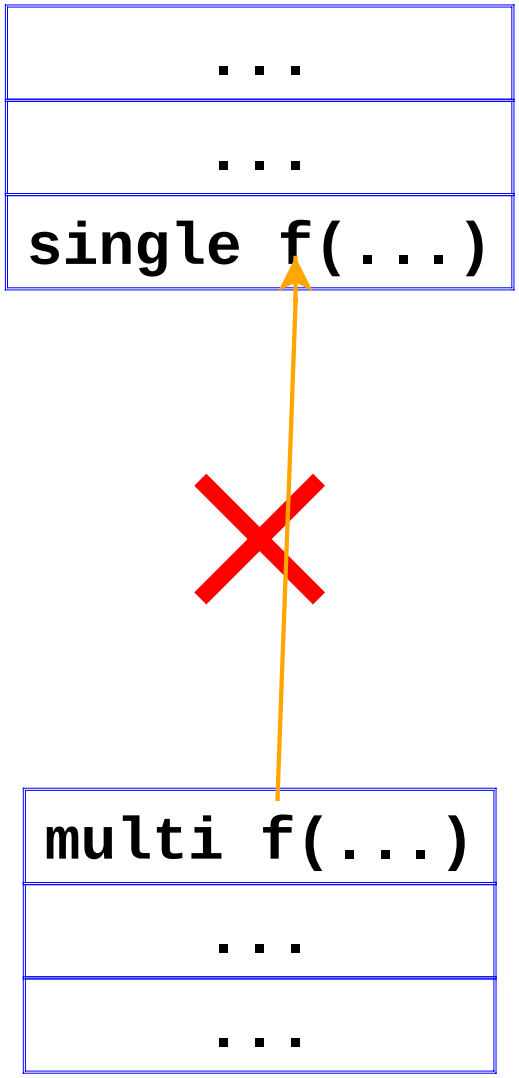
Constraint Propagation

Some components propagate constraints:



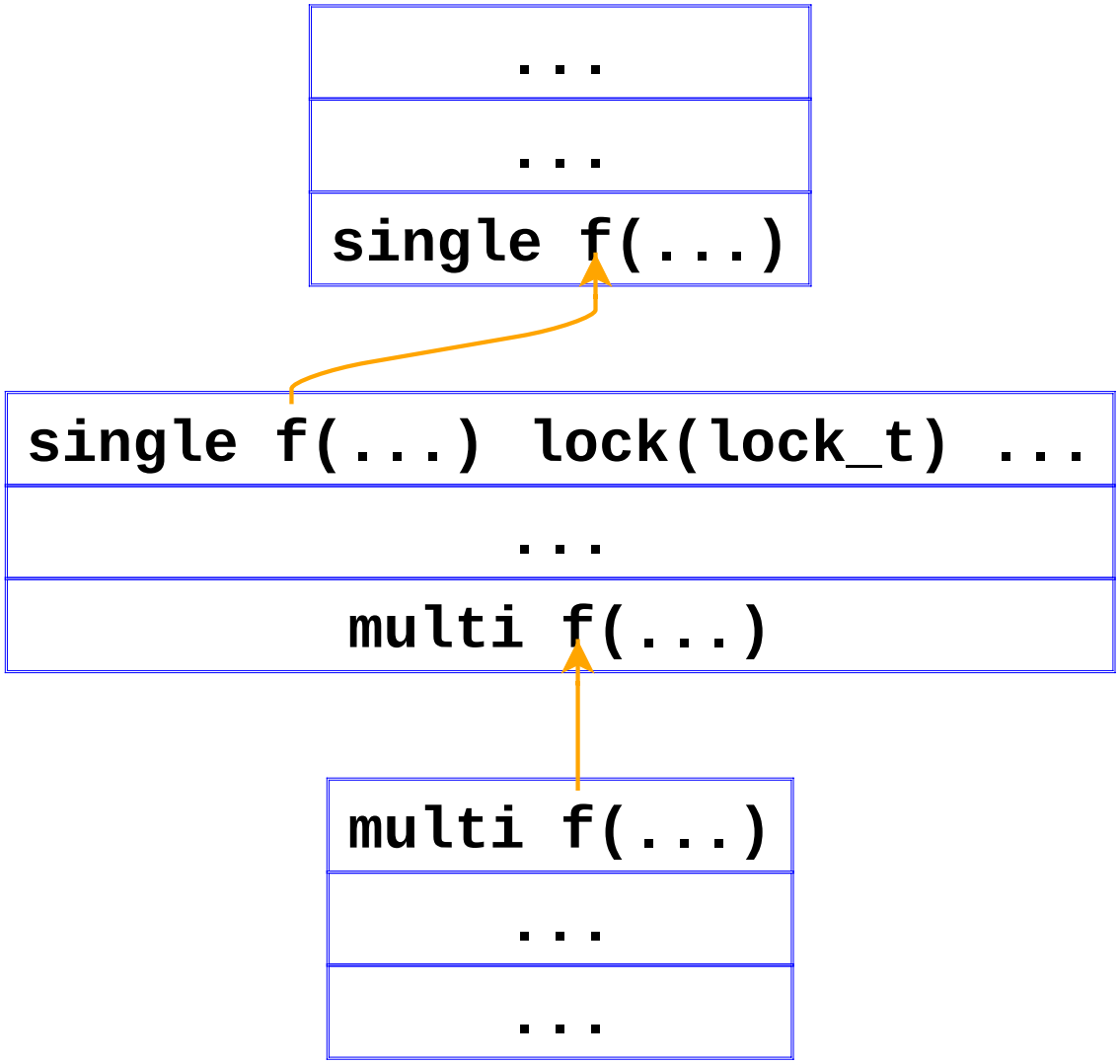
Inferring Adaptor Components

Automate mismatch repairs:



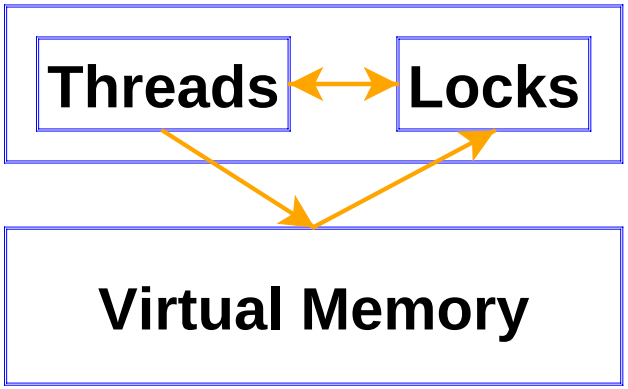
Inferring Adaptor Components

Automate mismatch repairs:



Ordering Component Initializations

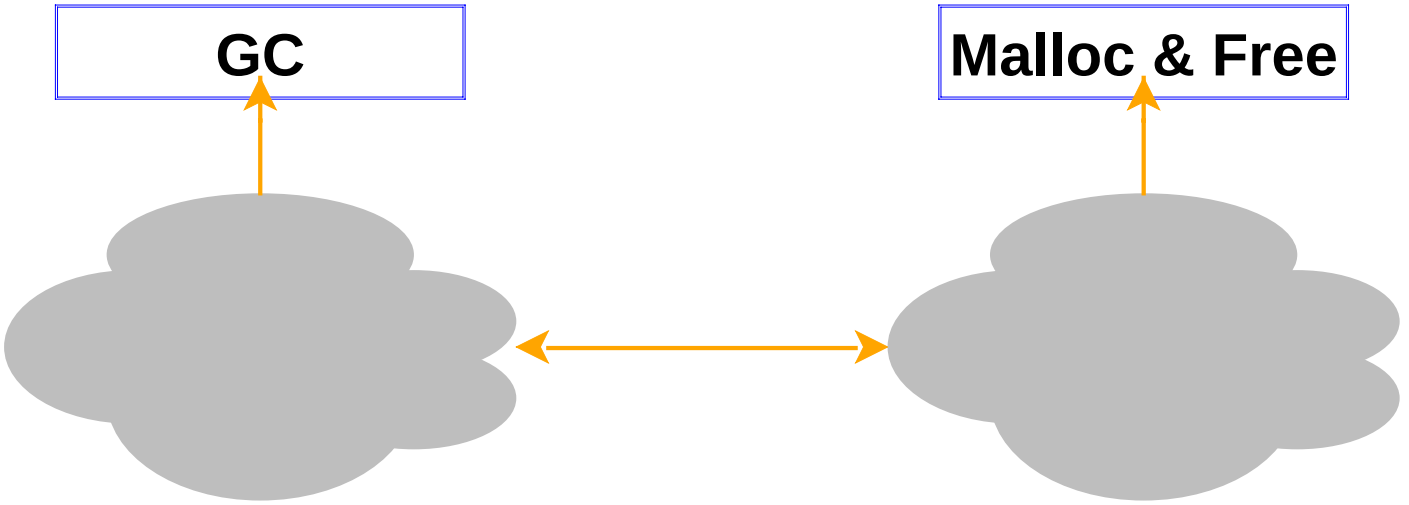
Auto-schedule component initialization:



⇒

import$_1$ ... import$_n$	
definition$_1$ **...** **definition$_m$**	**init-dep$_1$** **...** **init-dep$_p$**
export$_1$ ... export$_k$	

Memory Management



- Components used with GC must be GC-friendly
- Boundary between GC and **`malloc/free`** components may need adaptors

Constraint Challenges

Summary of constraint challenges:

- Extensible
- Automatable checking
- Automatable adaptation

Alchemy Outline

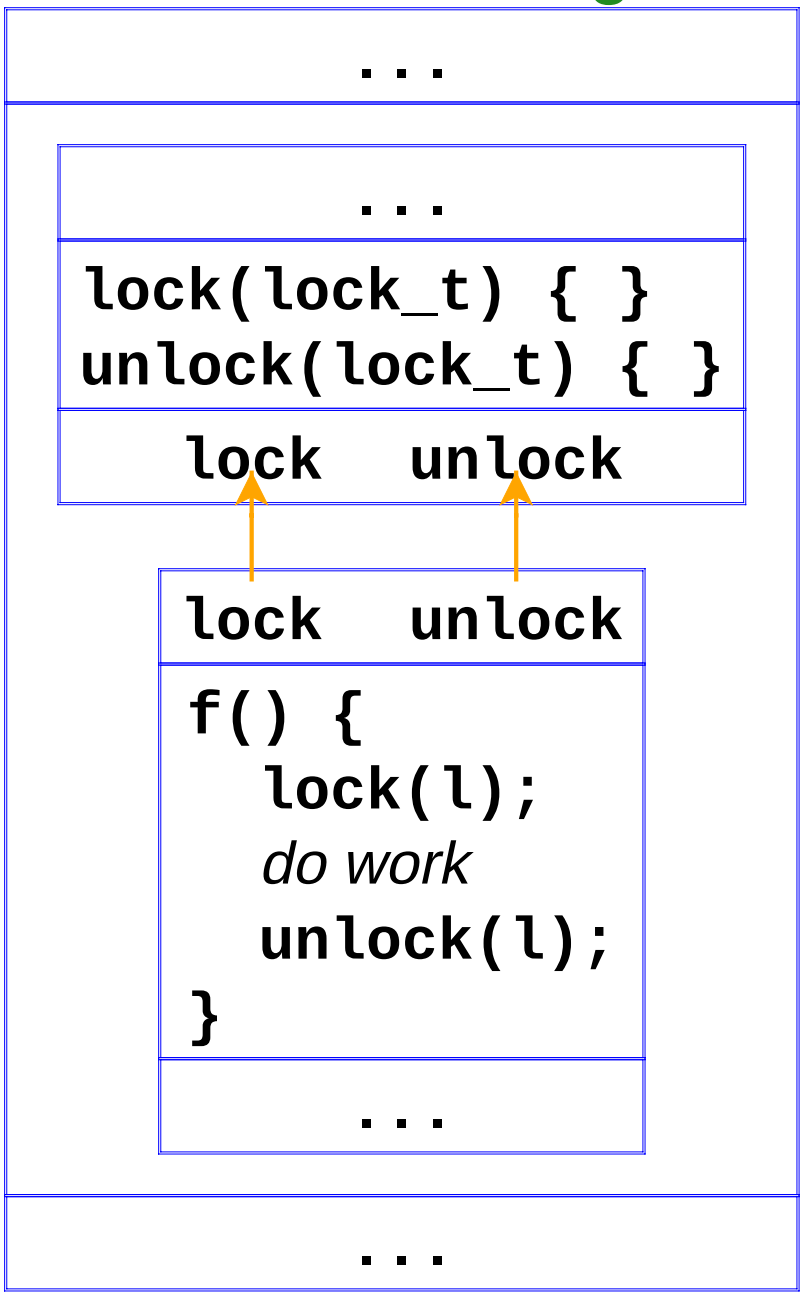
- **Language**
  - define components
  - link components
- **Types and constraints**
  - verify linking
  - infer adaptors
- **Performance**
  - zero componentization overhead
- **Applications**
  - The OSKit
  - Linux, ...

Performance

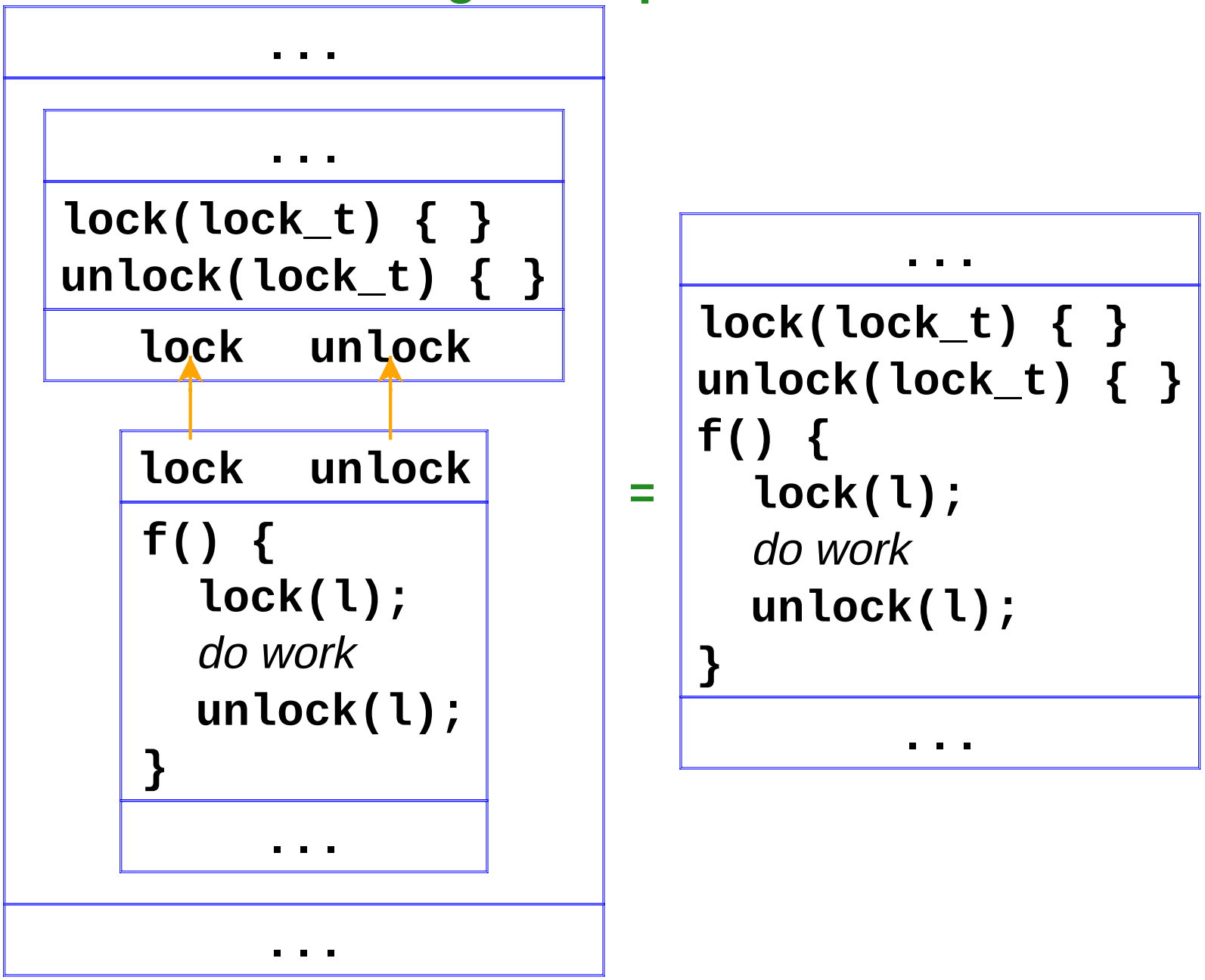
Performance goals:

- **+** Make aggressive componentization practical
- **-** Speed up existing code

Linking and Optimization



Linking and Optimization



Linking and Optimization

```
        ...
lock(lock_t) { }
unlock(lock_t) { }
f() {
  lock(l);
  do work
  unlock(l);
}
        ...
```

Linking and Optimization

```
          ...
lock(lock_t) { }
unlock(lock_t) { }
f() {
  lock(l);
  do work
  unlock(l);
}
          ...
```

=

```
    ...
f() {
   do work
}
    ...
```

Alchemy Outline

- **Language**
  - define components
  - link components
- **Types and constraints**
  - verify linking
  - infer adaptors
- **Performance**
  - zero componentization overhead
- **Applications**
  - The OSKit
  - Linux, ...

The OSKit

The OSKit: a set of components for systems software

- Extensive use of legacy code (e.g., Linux device drivers)
- 500 components when transmuted (estimate)
- Component sizes vary
  - Large: TCP/IP from FreeBSD, 18,000 lines
  - Small: Serial console, 200 lines
  - Smaller: Simple adaptors

The OSKit

The OSKit: a set of components for systems software

- `ld` works, but not well
  - Especially problematic for a new user
- COM works, but not well
  - Also problematic for a new user
  - Late discovery of errors
  - Reference counting difficult to maintain

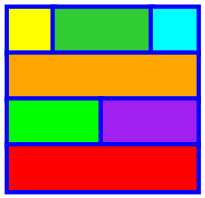
$\Rightarrow$ Alchemy

The Transmuted OSKit

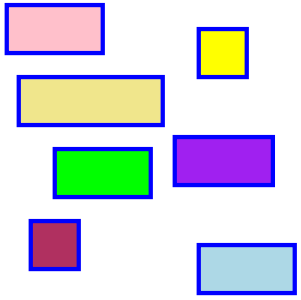
Transmuted OSKit = substrate for further PCES research

- Infrastructure for systems builders
- Bridge for more powerful analyses on systems code

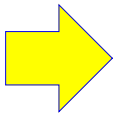
Linux

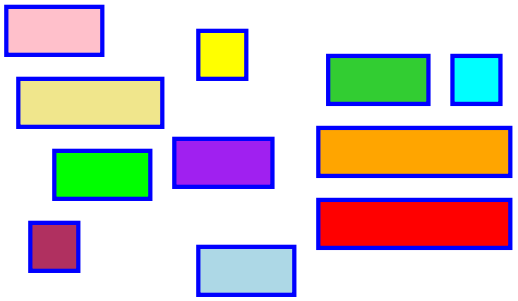


Linux

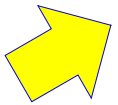


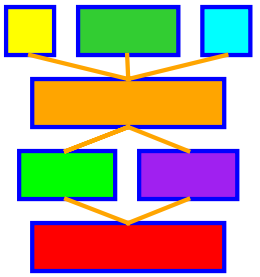
Current OSKit



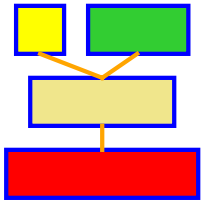


Expanded OSKit

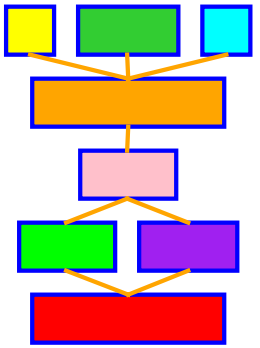




Regular Linux



Linux Lite



Linux RT

Alchemy

Component language and tools for low-level systems:

- Static linking makes analysis and optimization tractable
- Constraints for global consistency checking
- Result: robust components, bridge to further analyses