Systems Design and Implementation *I.4* – Naming in a Multiserver OS

System Architecture Group, SS 2009

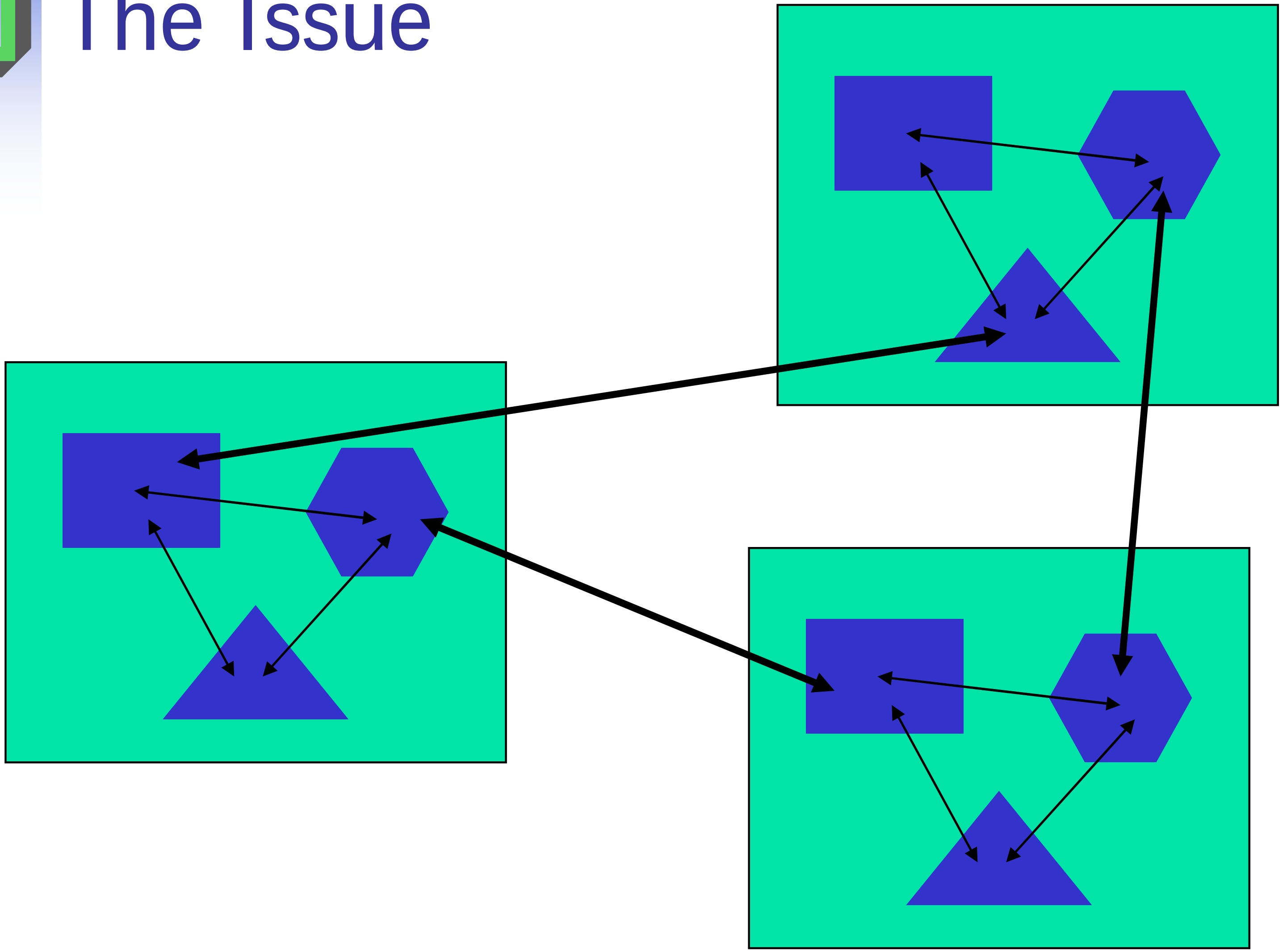
University of Karlsruhe

06.5.2009

Jan Stoess

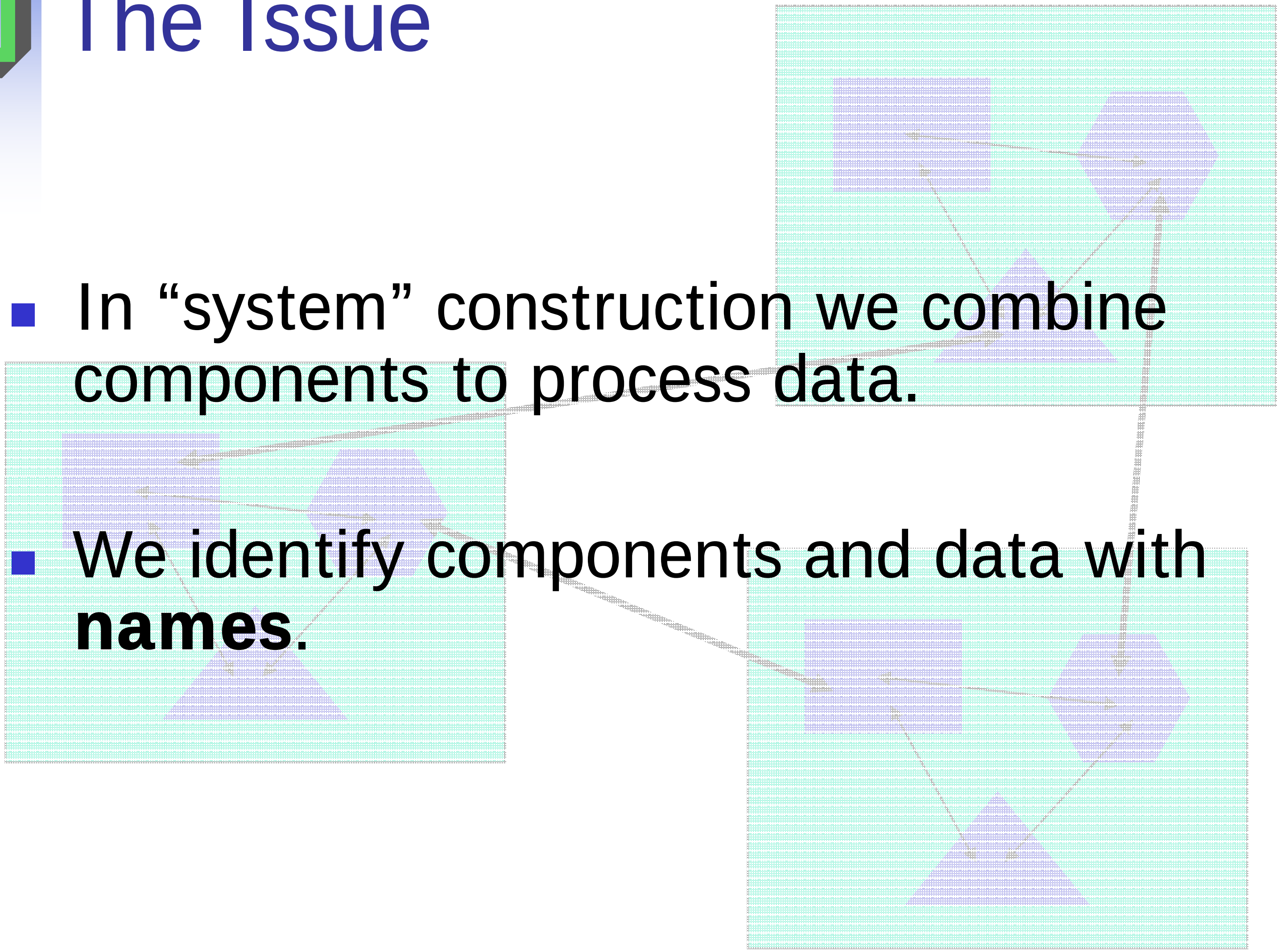
University of Karlsruhe

The Issue

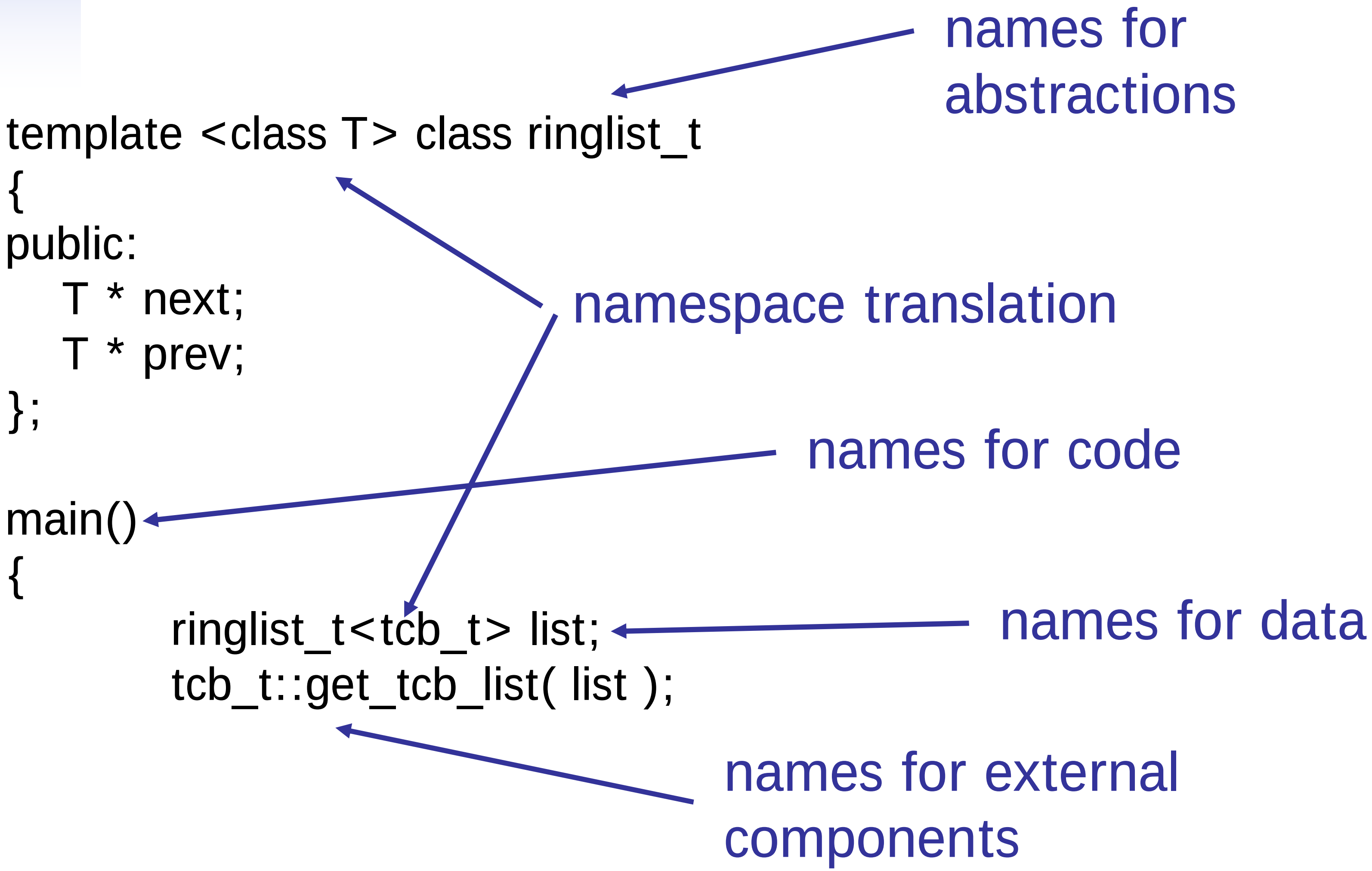


The Issue

- In “system” construction we combine components to process data.
- We identify components and data with **names**.



Names Example



```
template <class T> class ringlist_t
{
public:
    T * next;
    T * prev;
};

main()
{
        ringlist_t<tcb_t> list;
        tcb_t::get_tcb_list( list );
```

names for abstractions

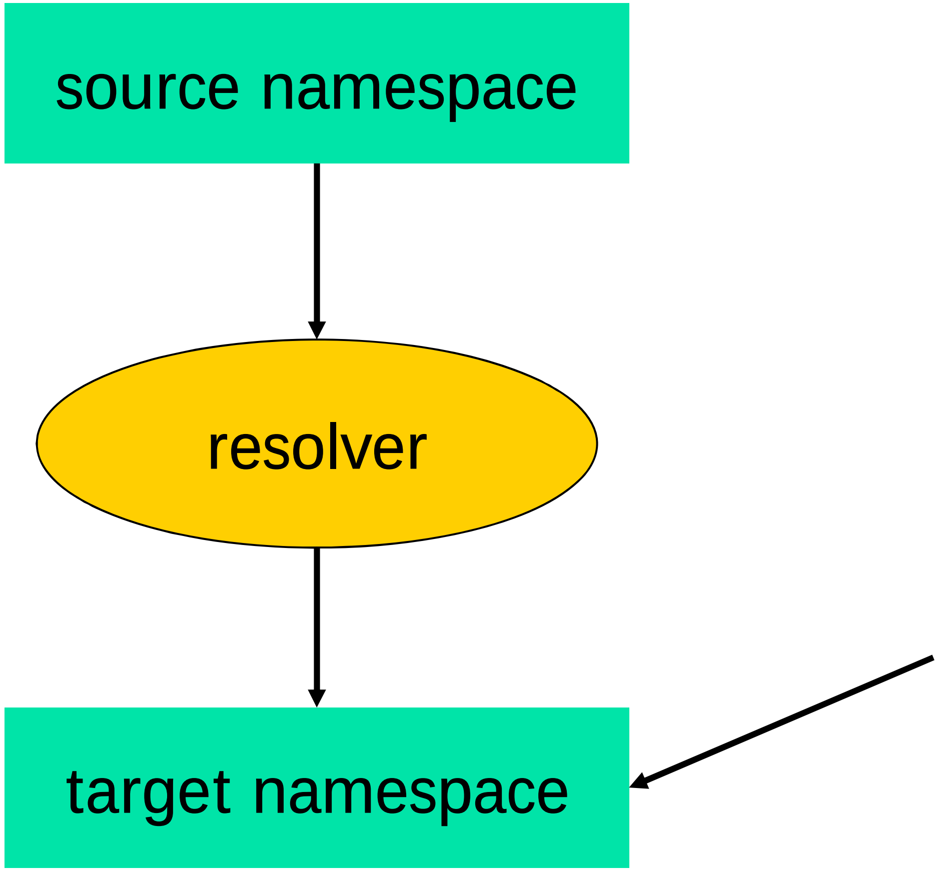
namespace translation

names for code

names for data

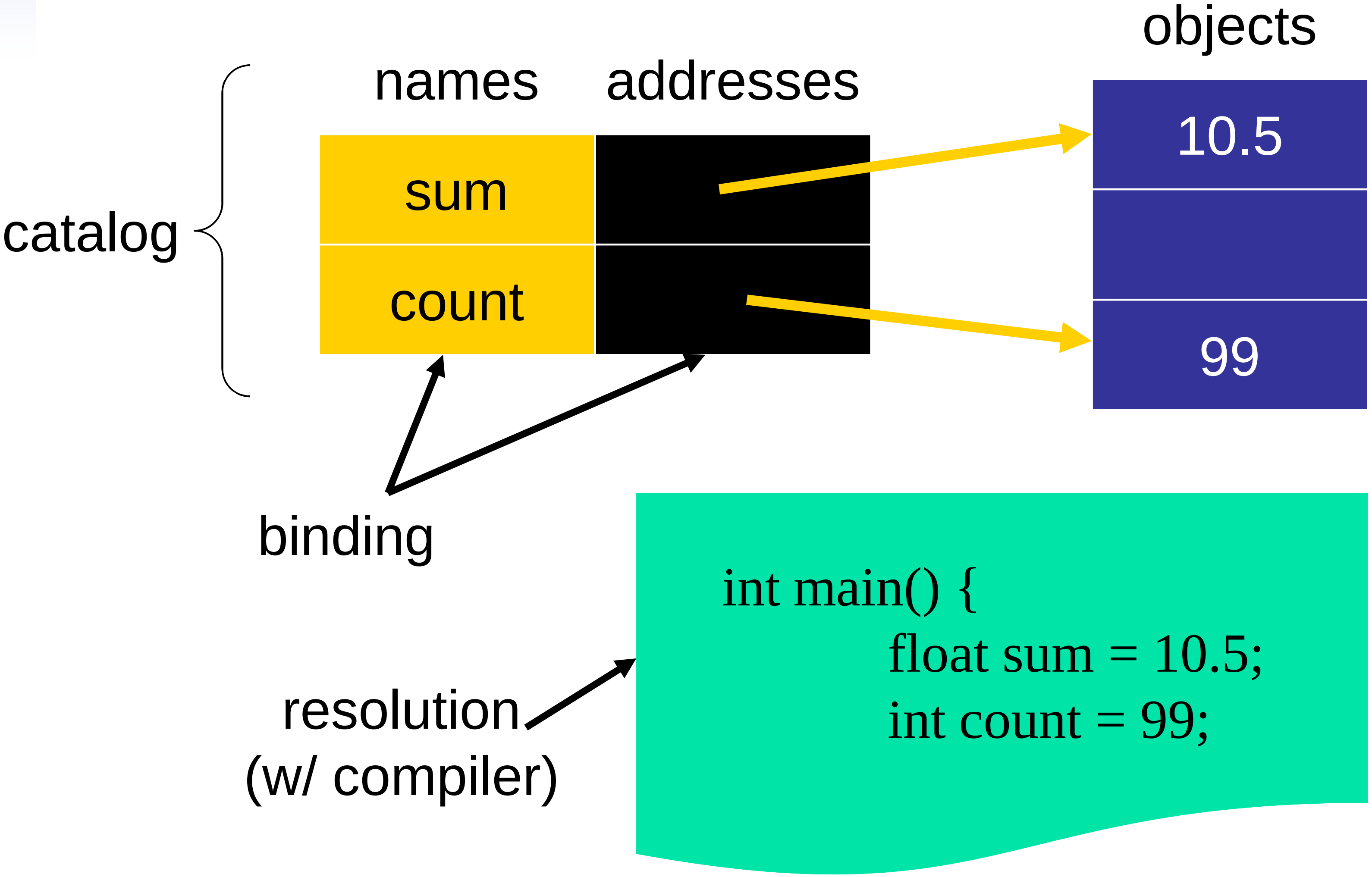
names for external components

Name Resolution



Often lower-level name, or closer to the object.

Naming Definitions



names

addresses

objects

catalog

sum

count

10.5

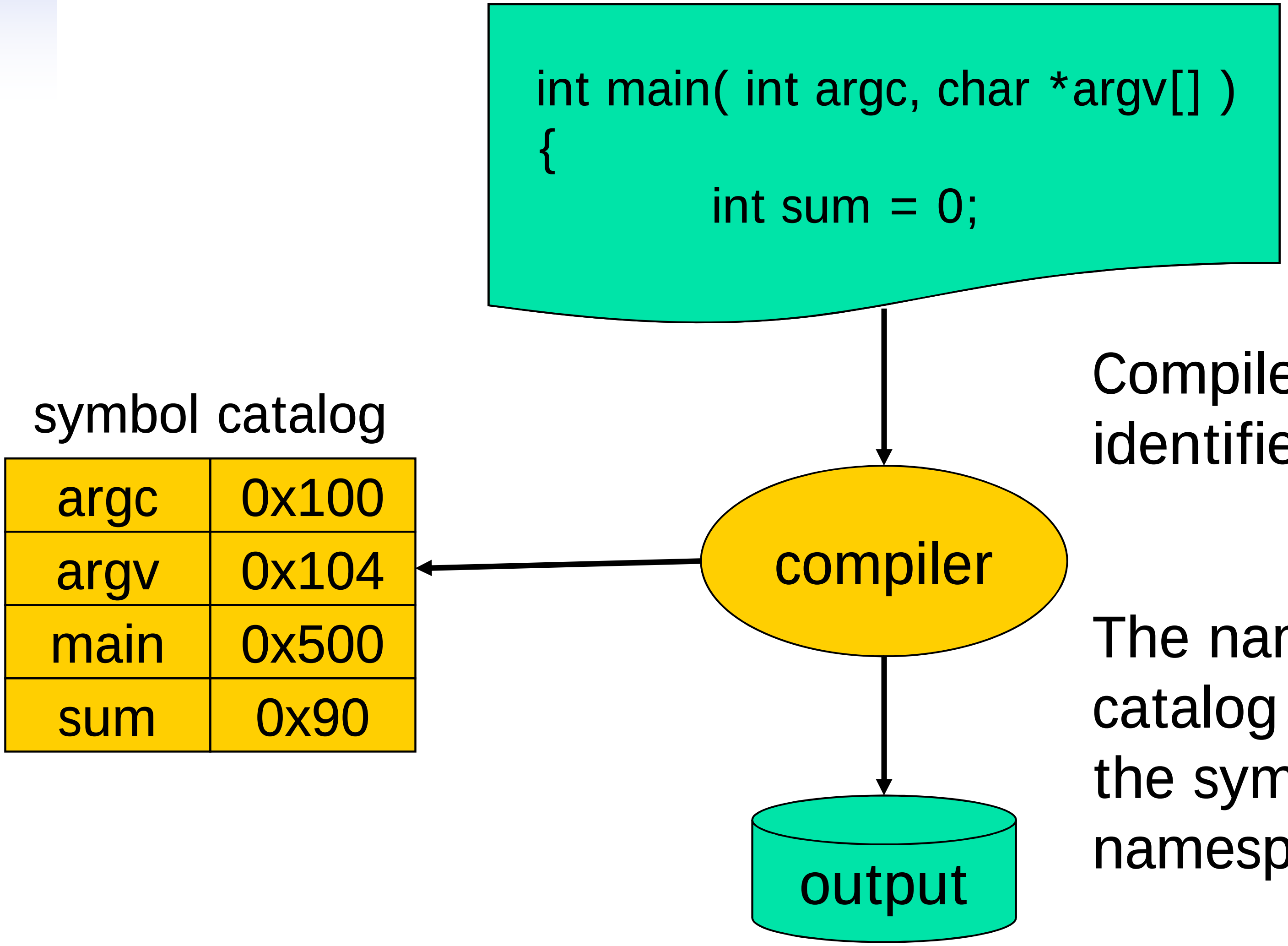
99

binding

resolution
(w/ compiler)

```
int main() {
        float sum = 10.5;
        int count = 99;
```

Closure



```
int main( int argc, char *argv[] )
{
        int sum = 0;
```

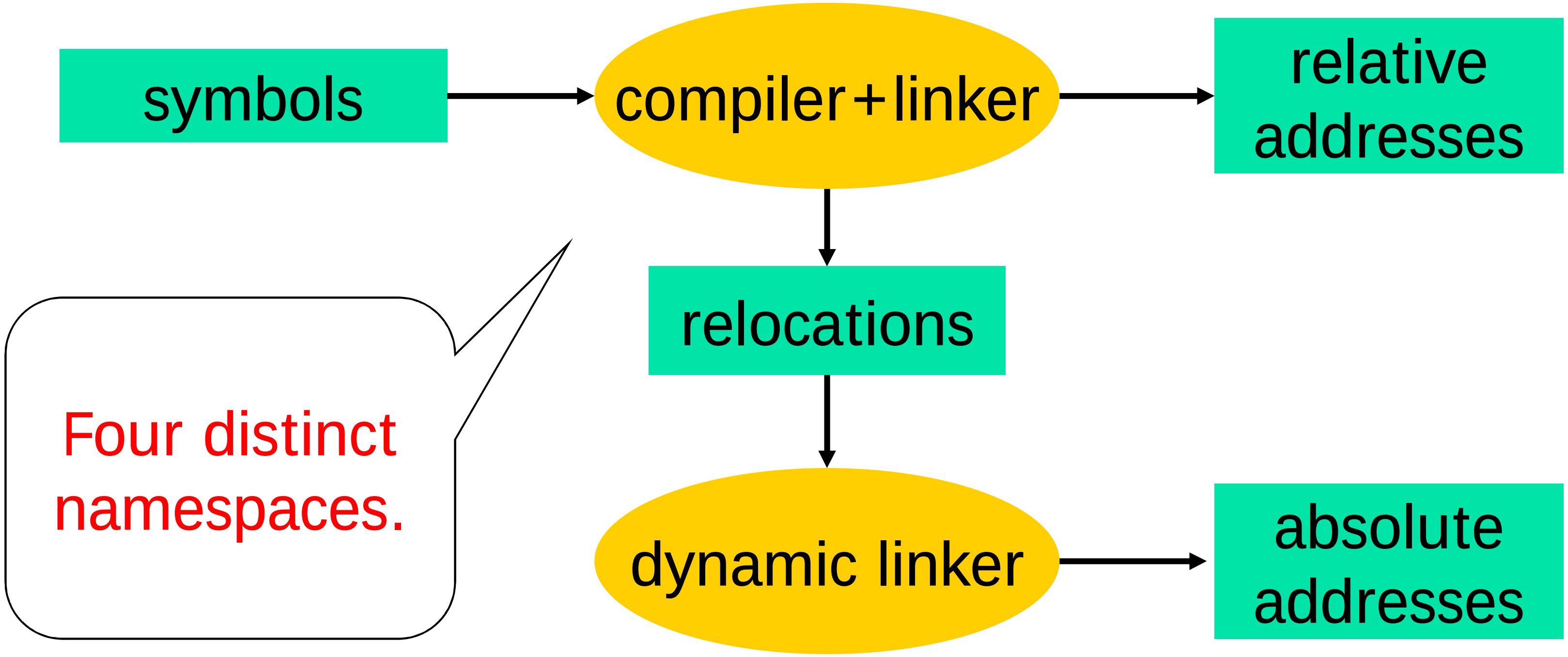
symbol catalog

argc	0x100
argv	0x104
main	0x500
sum	0x90

Compiler implicitly identifies catalog.

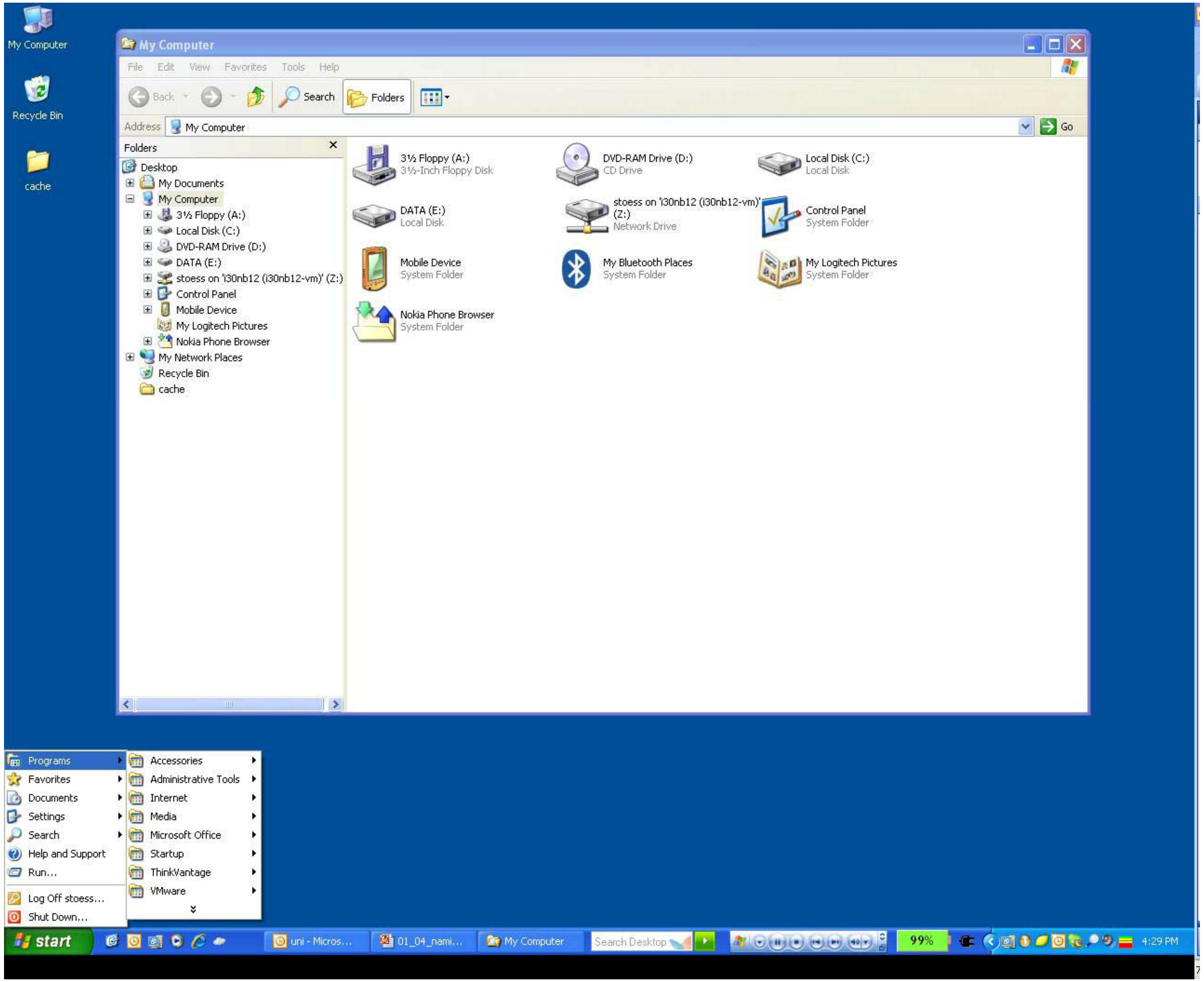
The name of the catalog is **outside** the symbol namespace.

Source-Code Name Translation

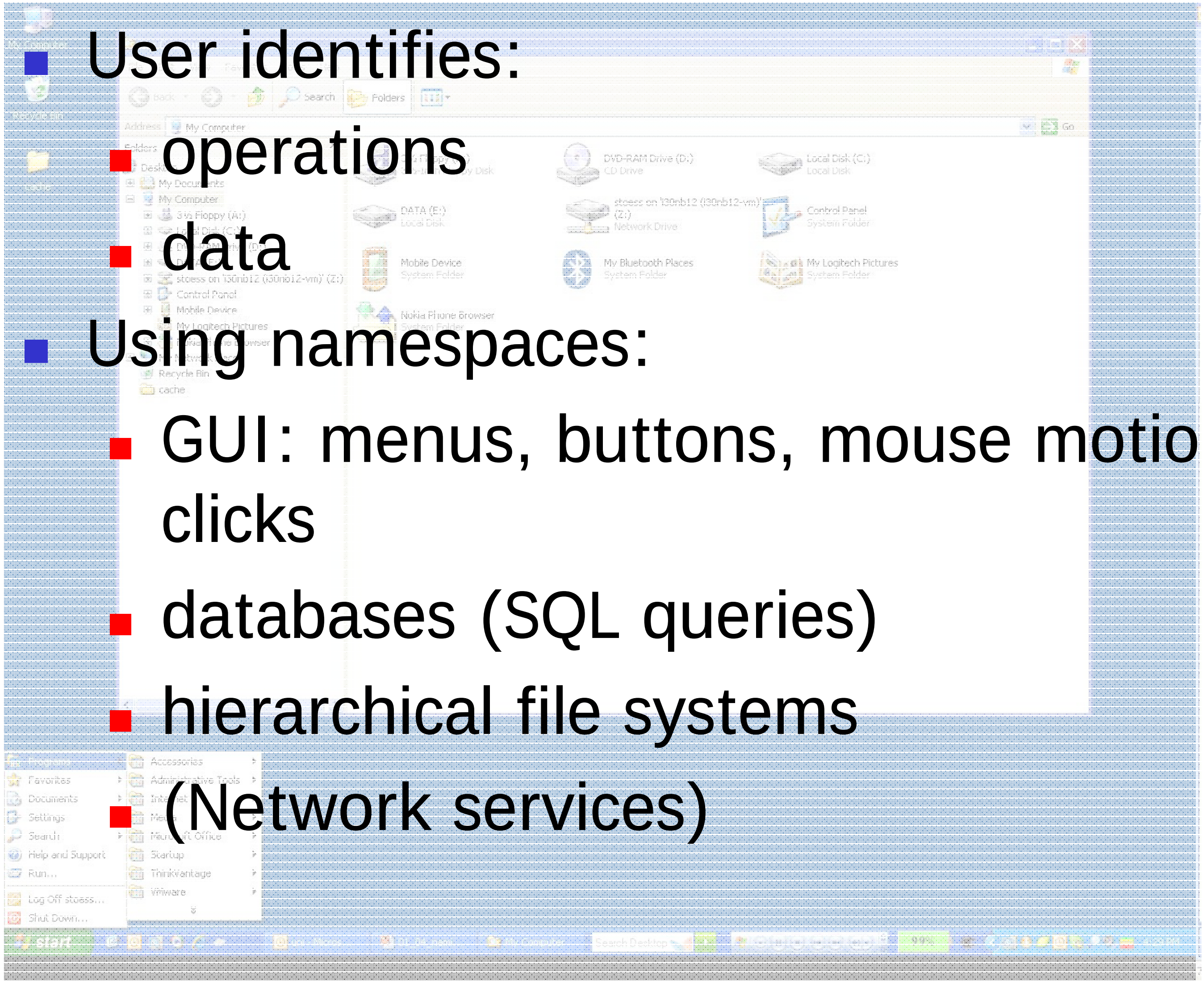


Four distinct namespaces.

User Run-Time Naming

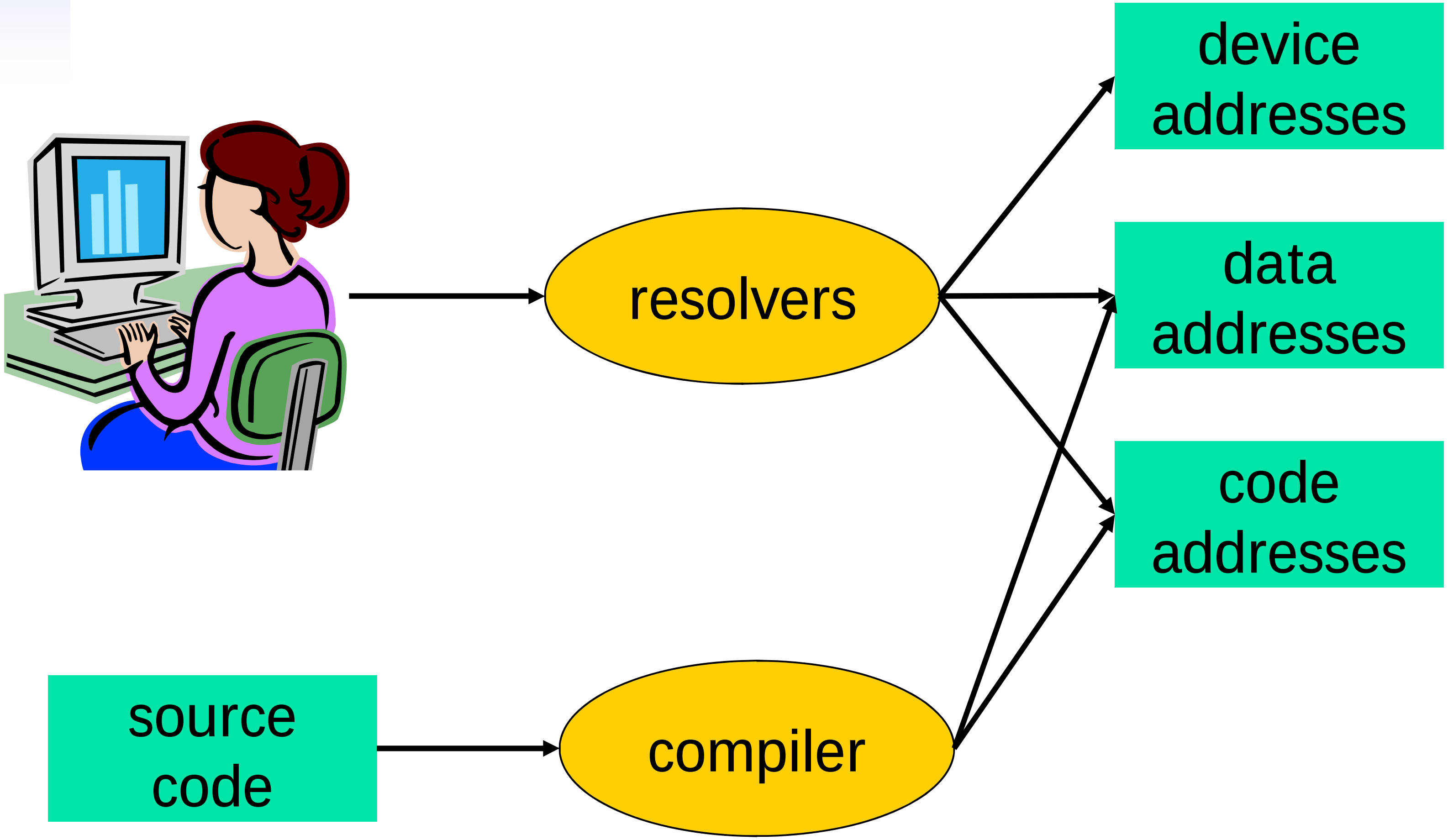


User Run-Time Naming

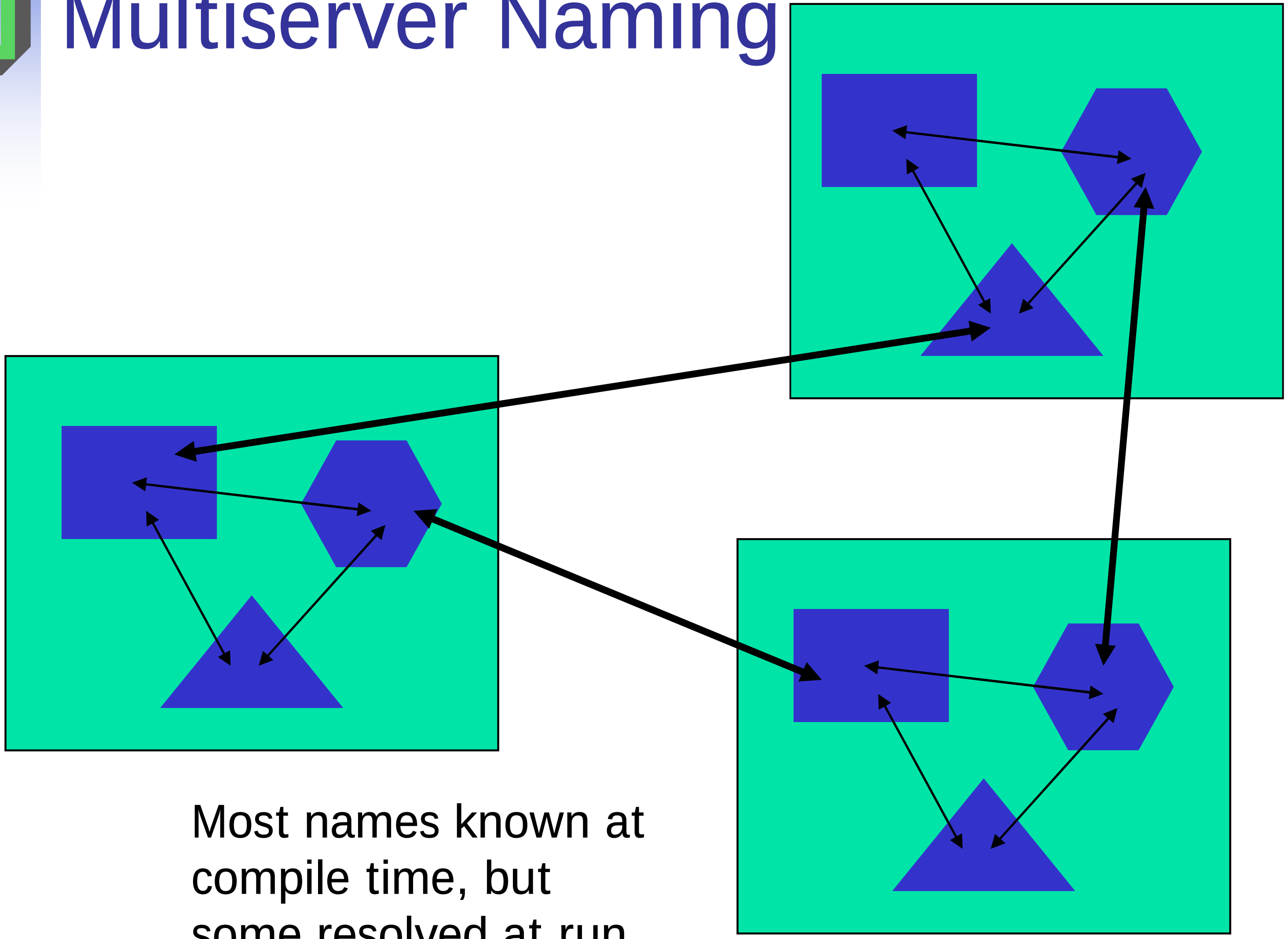


- User identifies:
  - operations
  - data
- Using namespaces:
  - GUI: menus, buttons, mouse motion + clicks
  - databases (SQL queries)
  - hierarchical file systems
  - (Network services)

User Run-Time Naming



Multiserver Naming



Most names known at compile time, but some resolved at run time.

Layered Naming

${HOME}/g001.jpg

/home/stoess/g001.jpg

/dev/hdb2/stoess/g001.jpg

disk2 :: partition 3 :: inode 40

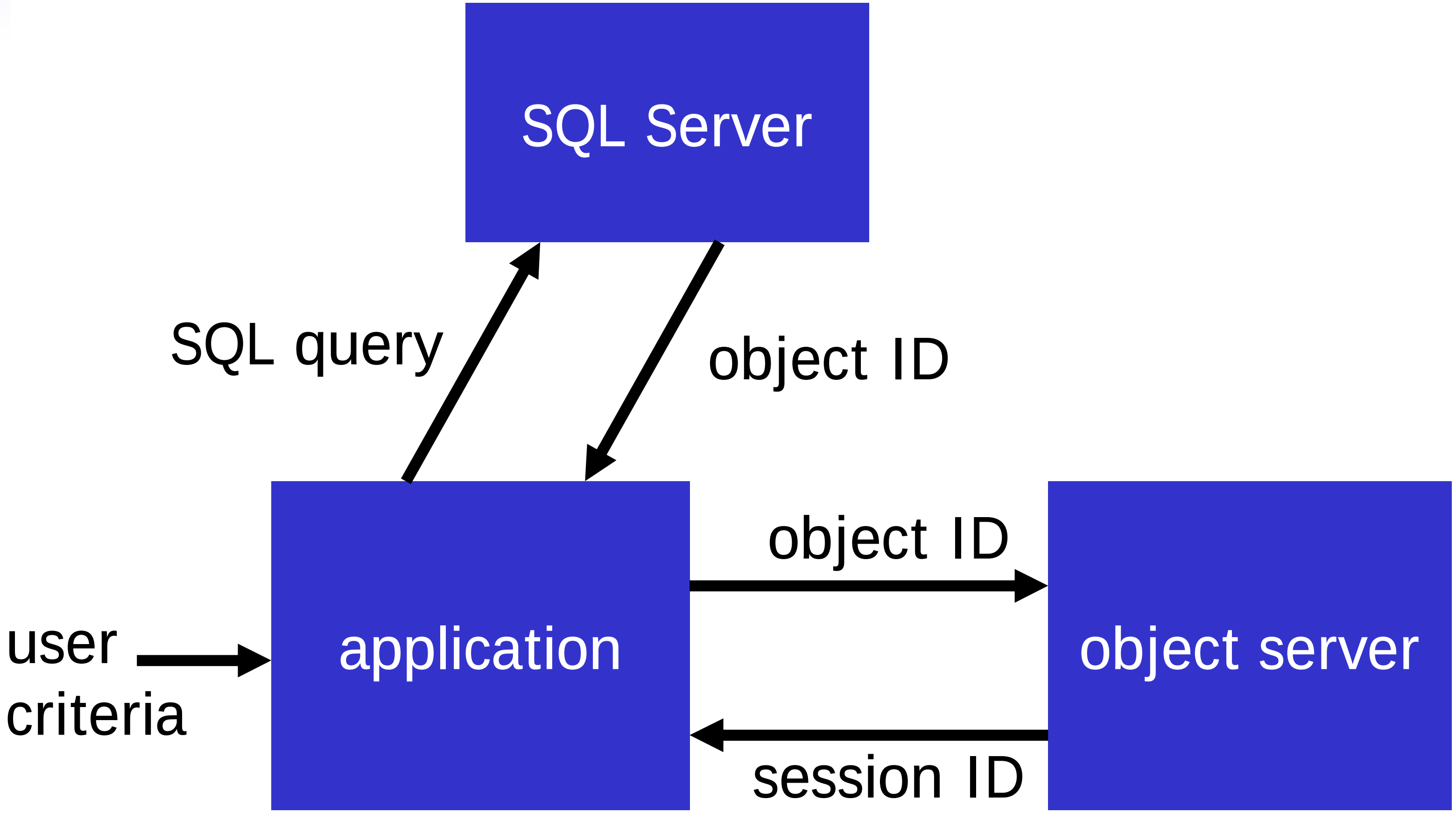
IDE address :: block offset

Naming as Indirection

- Why not name files by inode?
  - files could live at different inodes on different systems
  - two files may denote the same inode
  - inodes unpleasant to humans

- The concept: indirection
  - map a fixed namespace to a dynamic namespace
  - N:1 mapping possible
  - consistency problem

Indirection

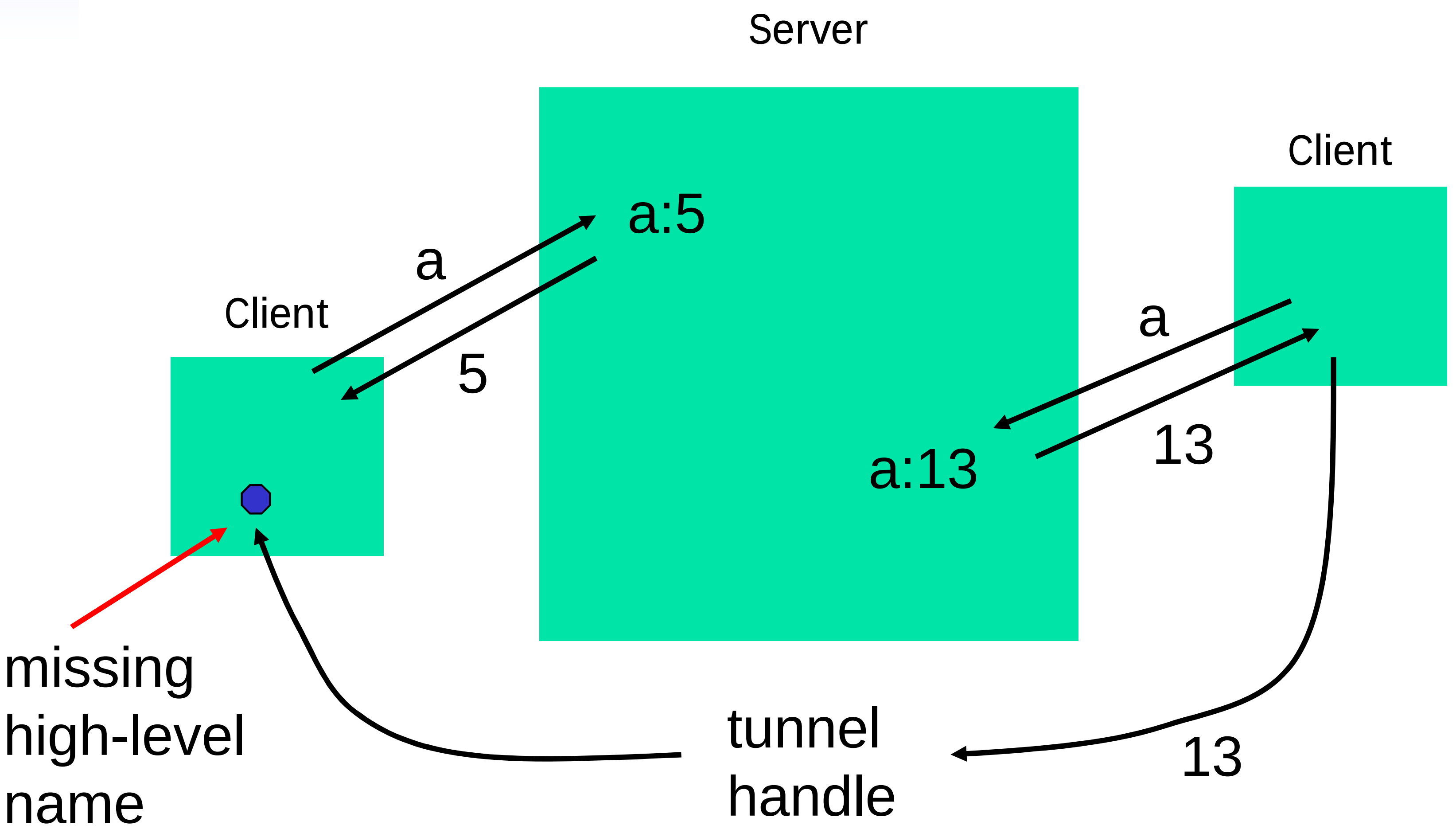


Problems with Indirection

- Unable to ensure that two people see the same object.

- Bindings are:
  - spatial
  - temporal

Context Sensitive Naming



Abstraction Level

- What should an API use for naming?
- Which abstraction level?

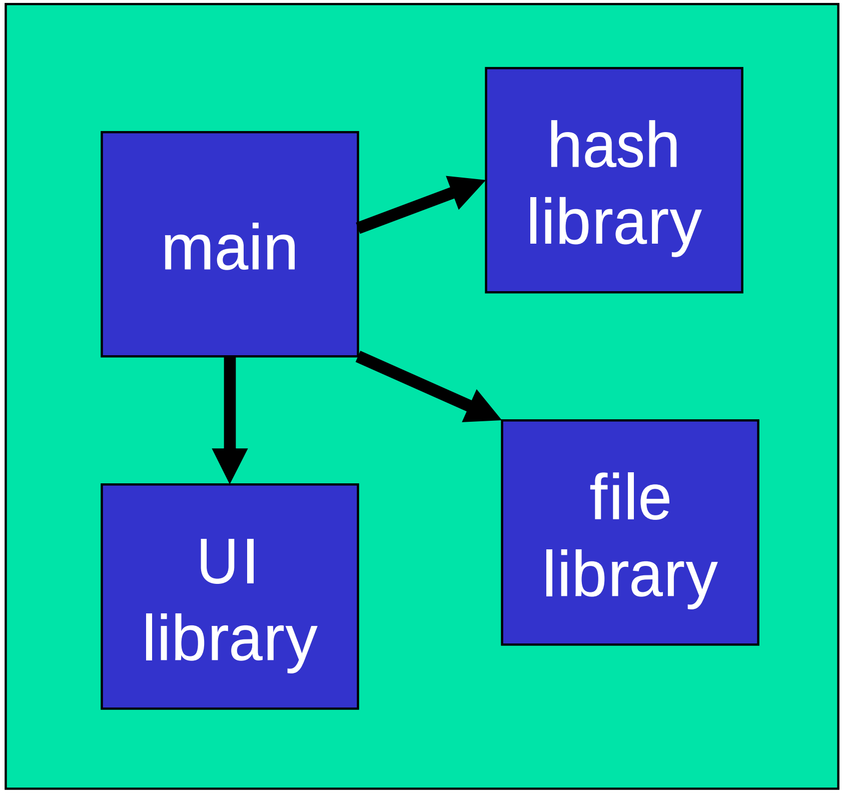
Binding / Catalog Creation

- When do we bind names?
  - compile time
  - run-time:
    - temporary
    - persistent

Resolution

- When do we resolve names?
  - compile time
  - dynamic binding (linking)
  - execution

Intra-Address Space Naming

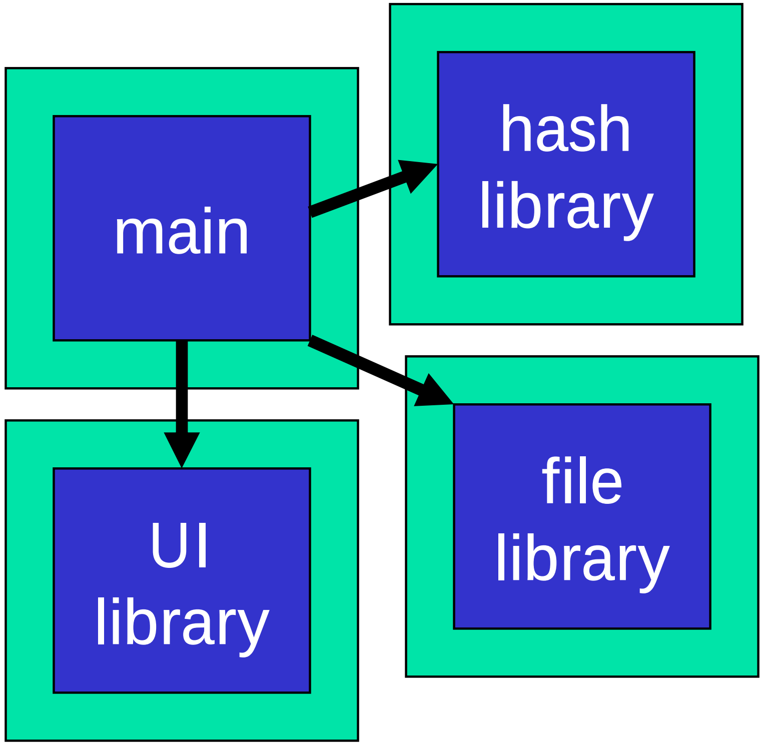


Naming: source code symbols, translate into addresses.

Protocol: function calls with pass-by-value and pass-by-reference data.

Resolution: compiler and linker.

Inter-Address Space Naming

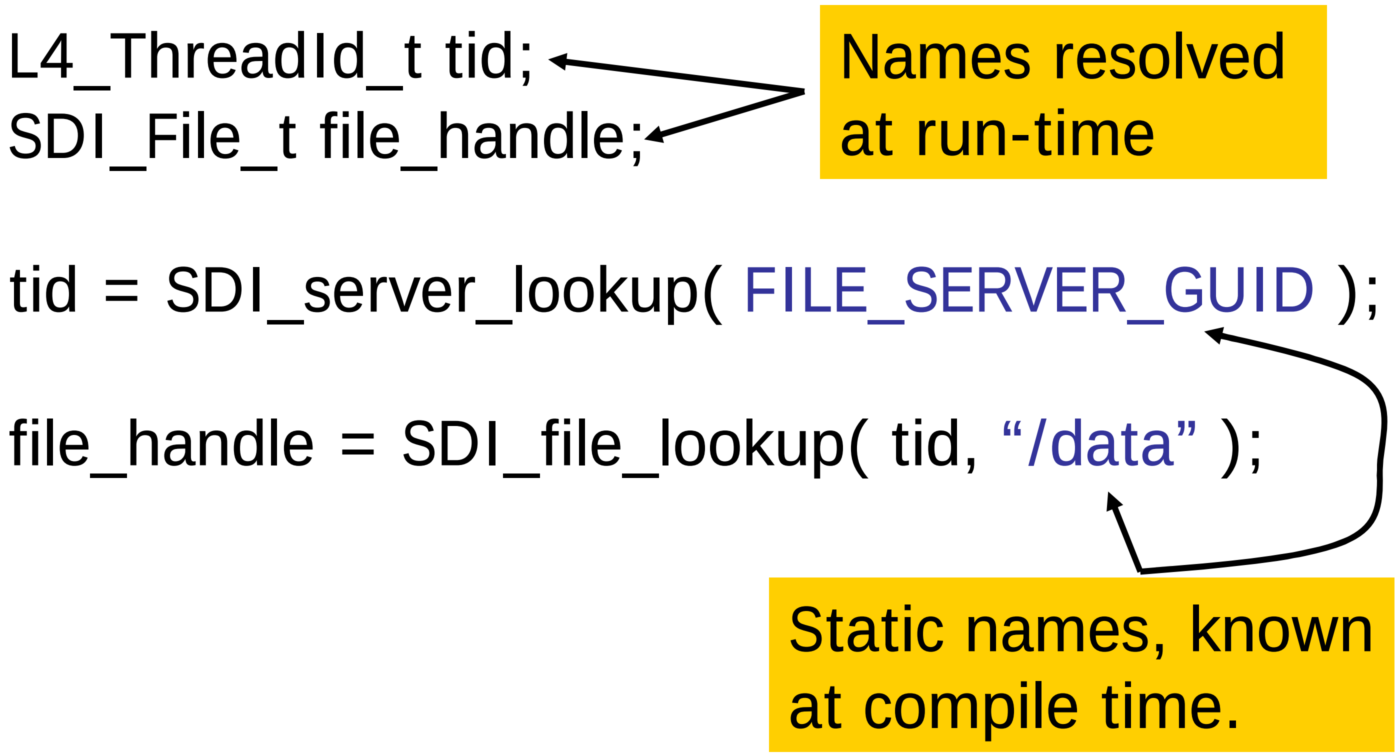


Naming: source code symbols, translated into handles at run-time.

Protocol: RPC with pass-by-value and pass-by-reference data.

Resolution: compiler, IPC, servers.

Name Use Example



```
L4_ThreadId_t tid;
SDI_File_t file_handle;

tid = SDI_server_lookup( FILE_SERVER_GUID );

file_handle = SDI_file_lookup( tid, “/data” );
```

Names resolved at run-time

Static names, known at compile time.

Catalog Maintenance

- Adding to the catalog
- Deleting from the catalog
- Enumerating the catalog
- Renaming entries (does renaming make sense?) - Provides atomic operation

- operations are inherently related to the target objects, and the closure

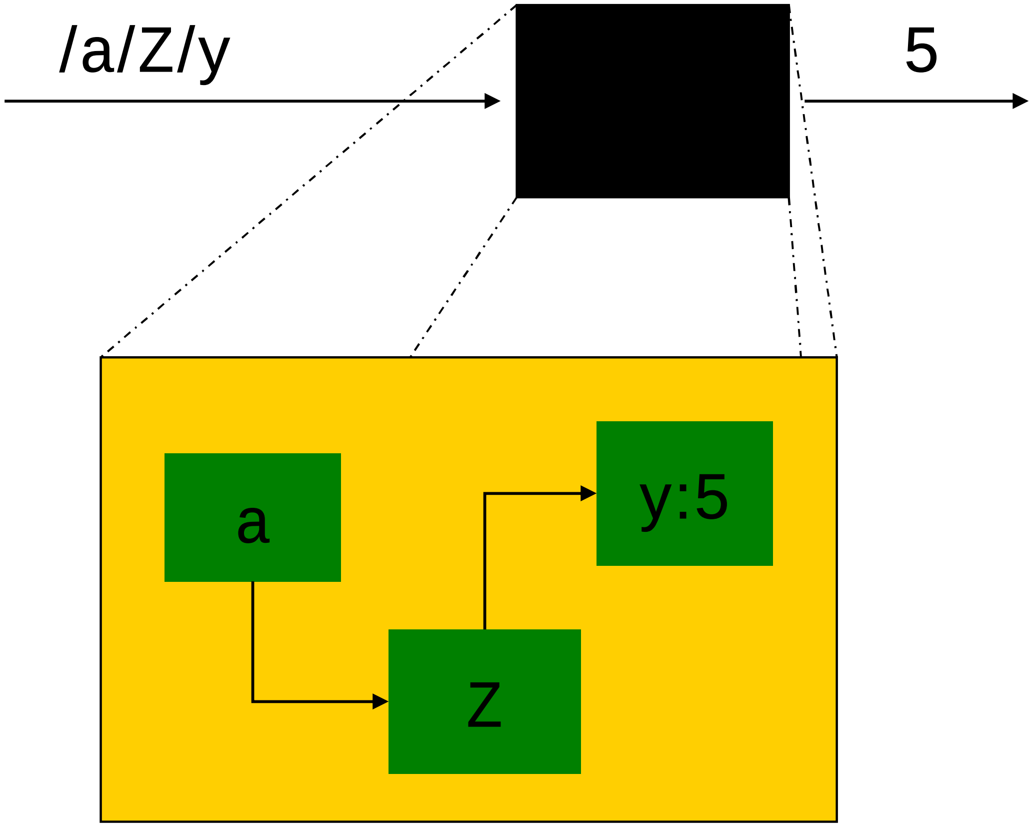
Namespaces

- Names are unique (within namespace)
- Names may have human meaning:
  - a file name
  - a sql query
- Names may have no human semantics:
  - exist solely to name an object
  - a memory address
  - an inode

How to Guarantee Name Uniqueness

- Central authority:
  - Active agent:
    - A process enforces uniqueness
  - Standards body:
    - ip addresses
- Distributed:
  - GUIDs
    - globally unique identifiers
    - statistically unique
- Combination:
  - Hostnames

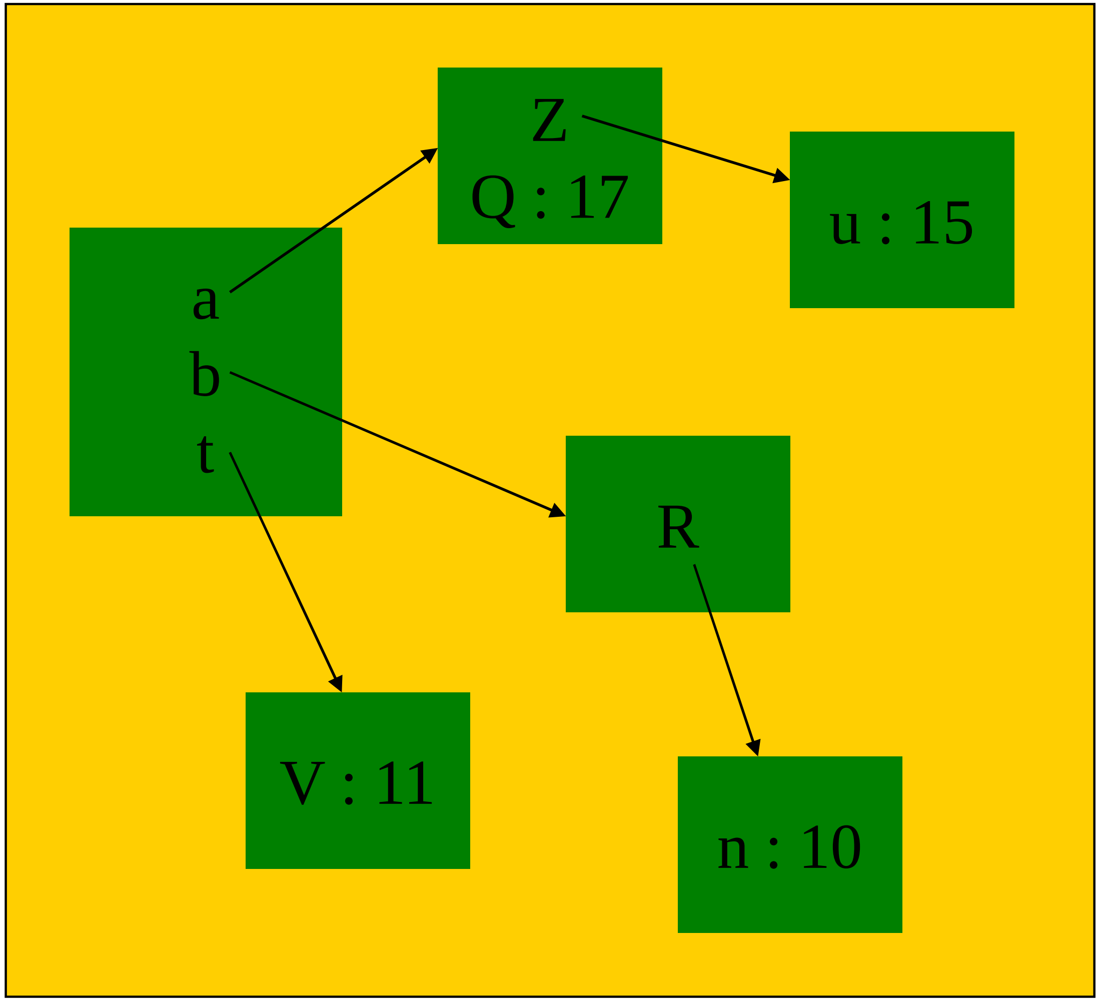
Name Resolution as a Black Box



Hierarchical Naming Implementations

/a/Z/u : 15
/b/R/n : 10
/a/Q : 17
/t/V : 11

↔



Hierarchical Naming

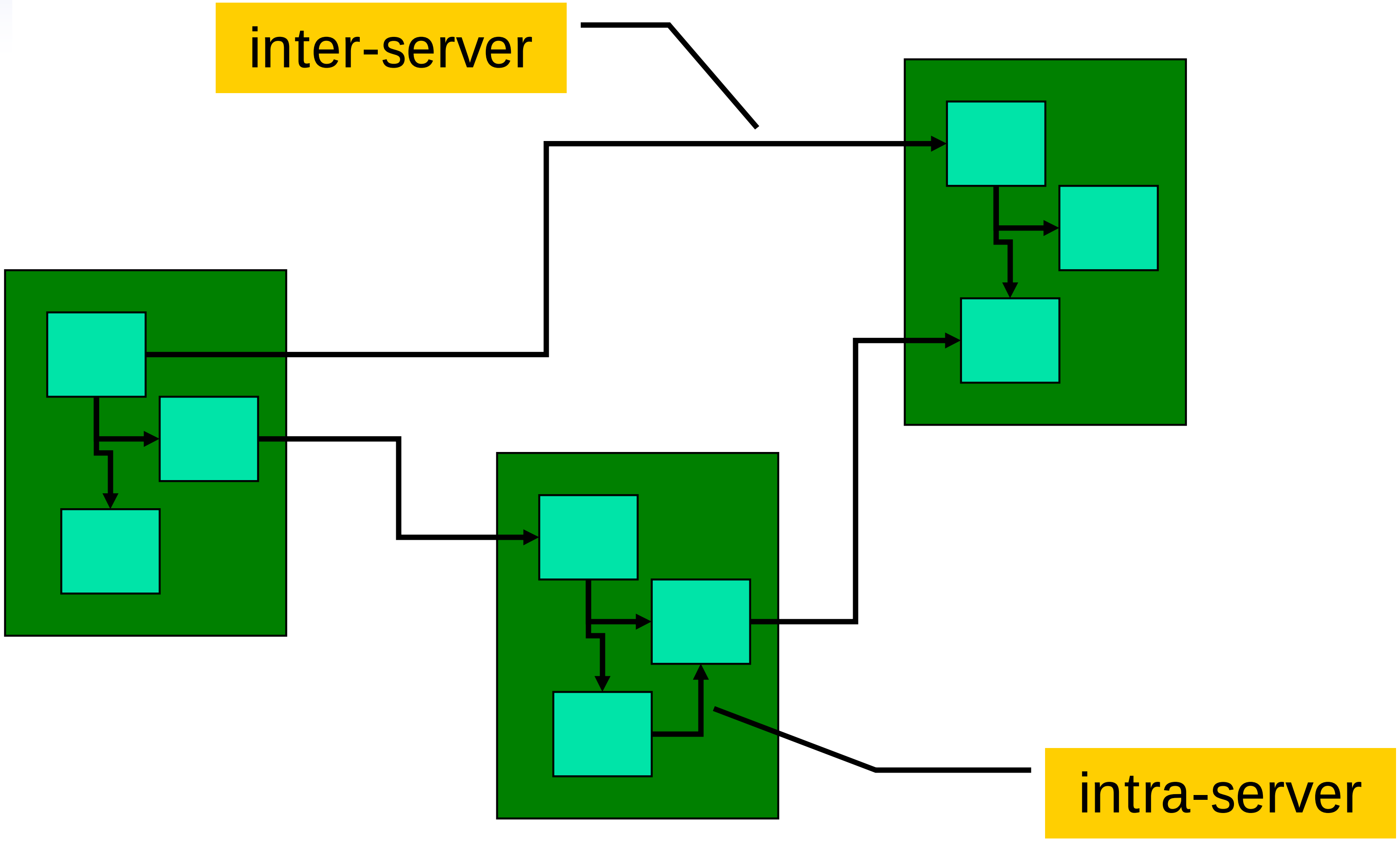
- Name contains names of catalogs leading to the target binding
  - Treats catalogs as distinct objects
- Impossible to name **root** catalog within name:
  - Root catalog implied by closure

Traditional Hierarchical Catalogs

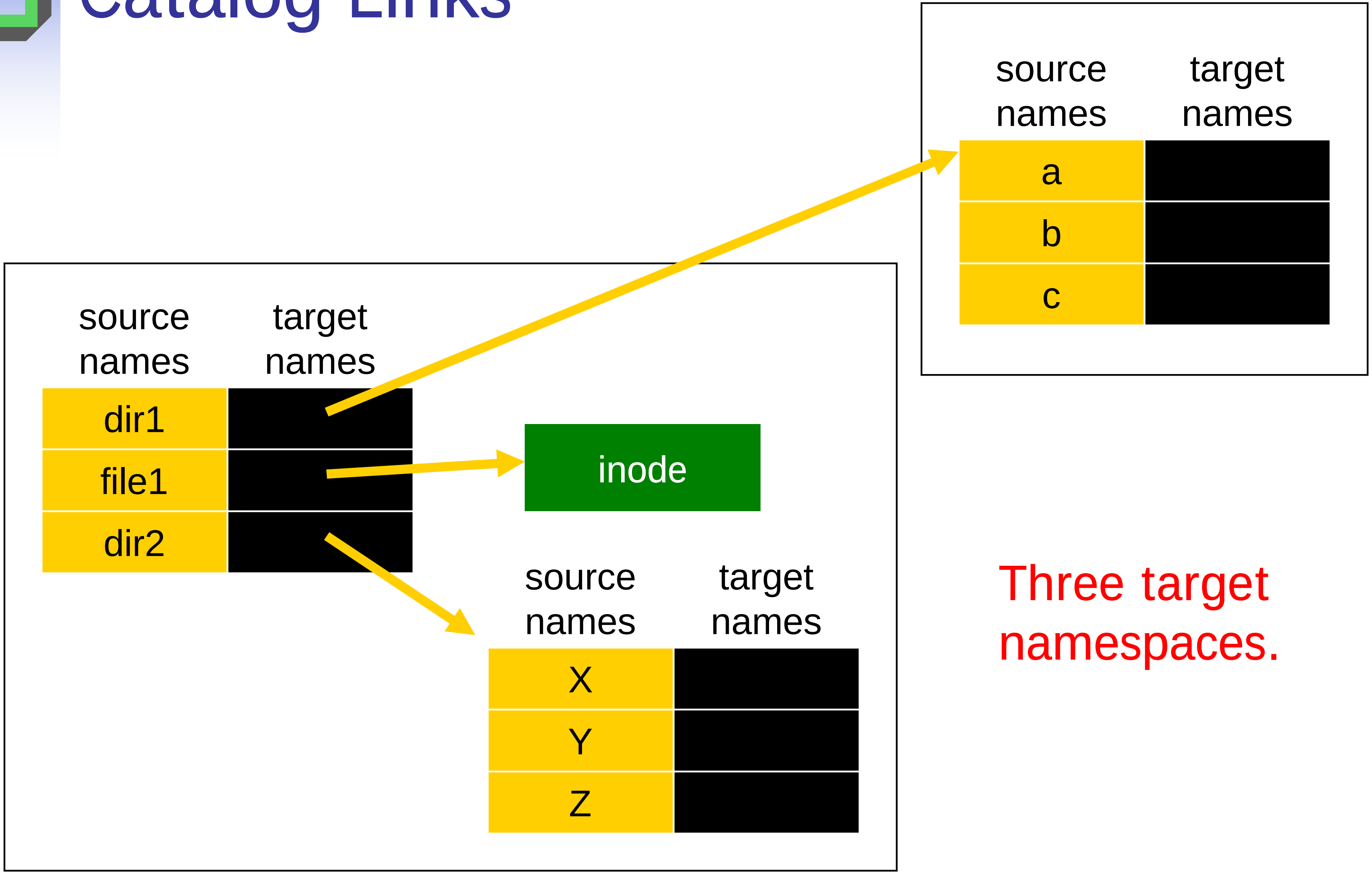
- Catalogs are distinct objects
  - Have their own properties

- Semantics of name are overloaded:
  - Security
  - Ownership
  - Location

Linked Hierarchical Naming



Catalog Links



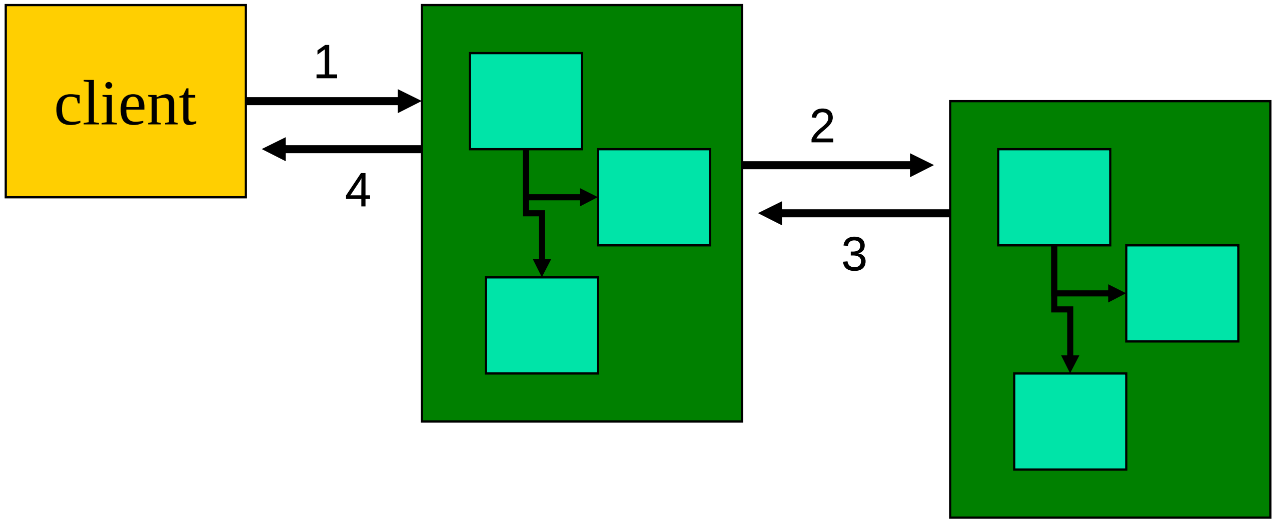
source names	target names
dir1	
file1	
dir2	

source names	target names
a	
b	
c	

source names	target names
X	
Y	
Z	

Three target namespaces.

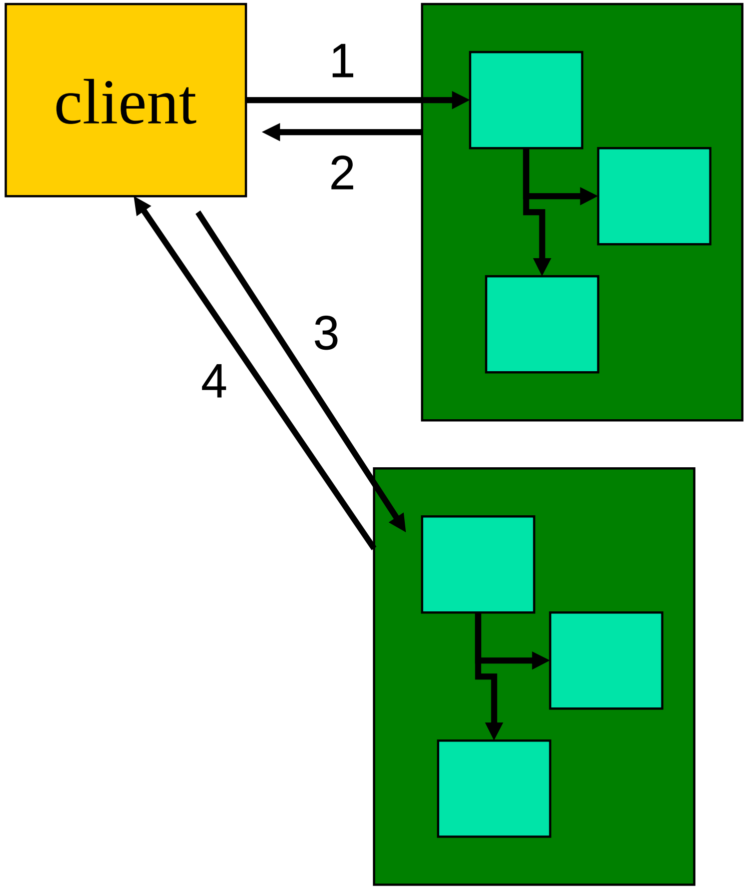
Recursive Inter-Server Links



Security vulnerabilities:

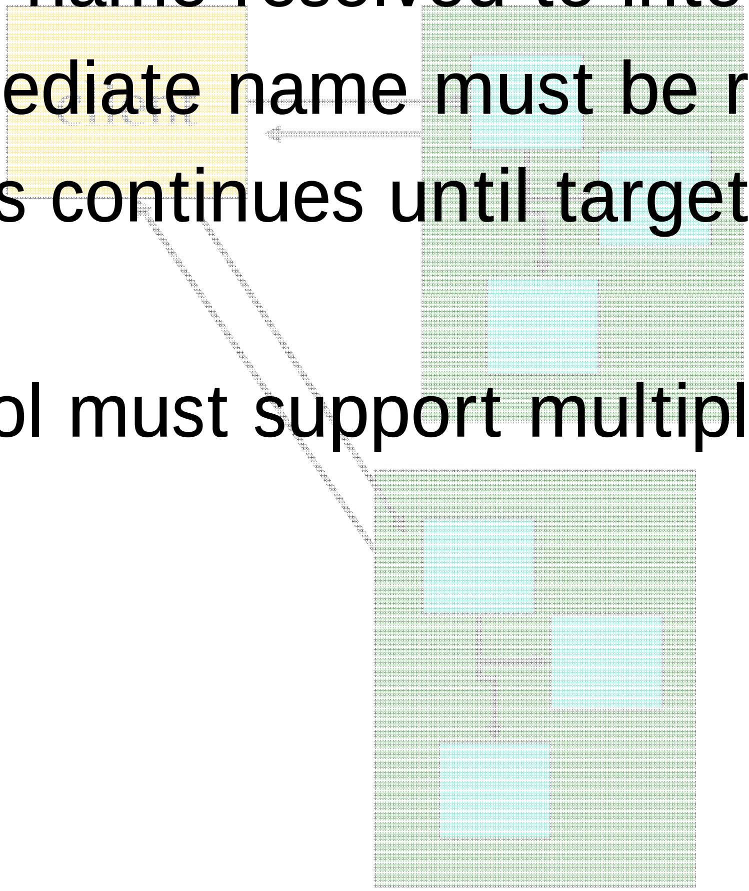
1. First server dependency on second server
2. Second server doesn’t know client identity

Iterative Inter-Server Links



Distributed Naming

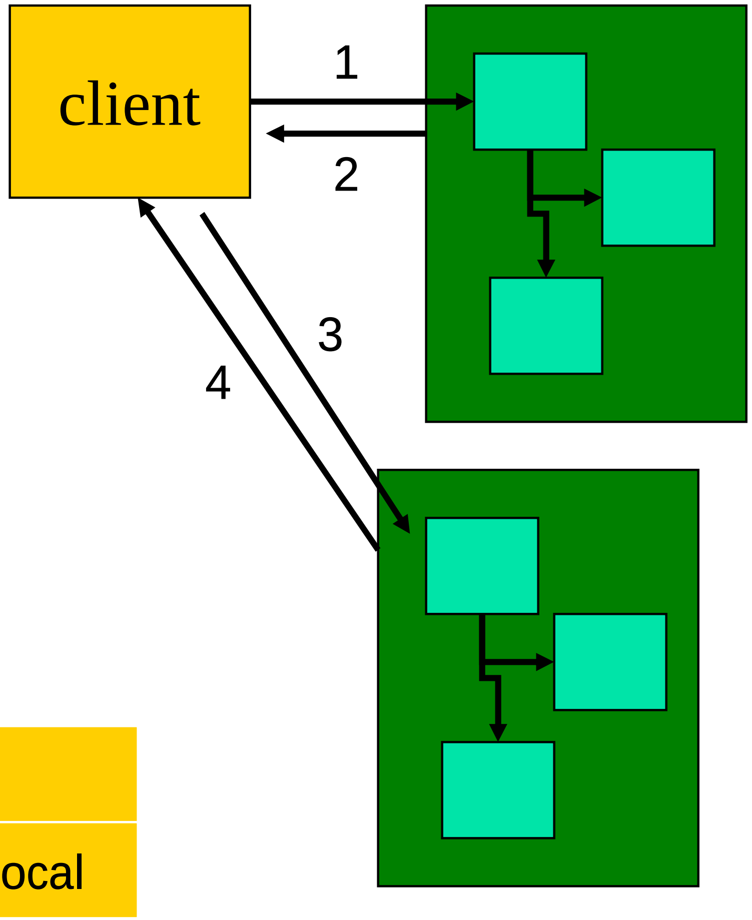
- Source name resolved to intermediate name.
- Intermediate name must be resolved.
- Process continues until target name resolved.
- Protocol must support multiple namespaces.



© 2009 University of Karlsruhe, System Architecture Group

Distributed Naming Performance

- Multiple IPC requests
- Answer: intermediate name caching
  - Name prefixes
  - Cache fairly static names



Prefix	Intermediate Name
/usr	TID 2, /export/usr
/usr/local	TID 5, /export/yoda/local

Distributed Naming Problems

- Consistency
  - Name cache out-of-date
  - Partial name change during resolution
  - For strict-consistency: verify name
- Possible to resolve a name binding that:
  - did not exist at start
  - does not exist at end

SDI Homework

What are your namespaces?

- Some of the namespaces to be implemented:
  - service names
  - interface names
  - file names
  - running task list
  - ...

Scope

- Compare the namespaces
  - what are their similarities?
  - what operations to support on their catalogs?
  - how are the names used?
  - should a namespace support distributed resolution?

Implementation

- Code reuse?
  - should you use the same namespace API for all namespaces?
  - example: hierarchical distributed namespace
    - source namespace: ASCII strings
    - target namespace: integers

(Does it make sense to use English for a namespace?)

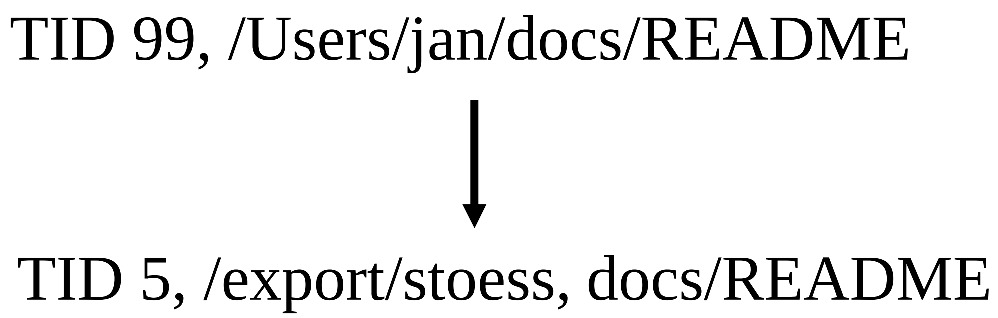
Integration

- Namespace integration?
  - if same namespace API for all namespaces ...
  - collect all namespaces into a single distributed, hierarchical namespace?

- If single, hierarchical namespace:
  - what is the target name?
    - object handles and TIDs in same namespace
    - how do you know which is which?
  - what interfaces does an object support?

Distributed Namespace

- If a distributed, hierarchical namespace:
  - must develop an iterative translation protocol
  - source name is translated into a target name which exists in a different catalog:



TID 99, /Users/jan/docs/README

TID 5, /export/stoess, docs/README

Your Assignment

- Design the appropriate IDL4 interfaces to support your namespaces
  - name resolution
  - catalog maintenance
  - **Use a distributed, hierarchical namespace scheme**

- Consider how the names will be used

More Remarks

Service Names

- Service: any L4 thread which publishes server-type functionality.
- Namespace: L4 thread IDs
  - We want to allocate and map thread IDs to services dynamically
  - Use names for indirection
- Clients know service names at compile time
  - We know we want to connect to a file server

Service Catalog

- How is service catalog named?
  - The service catalog is itself a service
  - Thus unable to name within service namespace
- How do clients name the service server?
  - Implied by closure
  - Convention can choose an implied name
    - Contact a specific server (a reserved thread ID)
    - Or map shared page in everyone’s address space

Operations on Service Catalog

- Resolve name
- Add binding
- Delete binding
- Rename binding?
- Enumerate bindings?

Interface Names

- We generally want to negotiate an interface with the server
- Interface names known at compile time
- For us, servers know which interfaces they support
  - Service catalog/semantics built at compile time
- An interface name maps to a set of handler functions within the server
  - Permit a server to support multiple interfaces per server thread
  - Use IDL4 inheritance

Interface Names

- We need an interface to negotiate interfaces
    - the name would be outside the naming system
    - must use closure to choose a default interface
        - convention may choose interface 0

Interface Negotiation

- Send an IPC to interface 0 of a server
- Query:

  support interface X on object A?

File Names

- Names created dynamically
- Names translate into a session handle as seen by the client
  - More efficient than typical text file name
  - Server may associate state with session handle
  - Session handle associated with an access interface
- The session handle maps to disk blocks in the server
- Tiered namespaces

Running Task Names

- What is the name of a task?
- How do you ensure uniqueness in the source namespace?

- Traditional procfs uses the PID as the source namespace.

Thursday

- Debugging on L4
- Takes Place in R149 50.34