μ-Kernel Construction (11)

Security

Is your system secure?

Security: A condition that results from the establishment and maintenance of **protective measures** that ensure a state of inviolability from hostile acts or influences. [Wikipedia]

Security Defined by Policy

- Examples
  - All users have access to all objects
  - Physical access to servers is forbidden
  - Users only have access to their own files
  - Users have access to their own files, group access files, and public files (UNIX)

Security Policy

- Specifies who has what type of access to which resources

Authentication

Authorization

All Access is via IPC

- What microkernel mechanisms are needed for security?
  - How do we authenticate?
  - How do we perform authorization?
  - How do we implement arbitrary security policies?
  - How do we enforce arbitrary security policies?

Authentication

- Unforgeable thread identifiers
  - Thread ID of sender returned by kernel
  - Thread identifiers can be mapped to
    - Tasks
    - Users
    - Groups
    - Machines
    - Domains
  - Authentication is outside the microkernel – any policy can be implemented

Authorization

- Servers implement objects; clients access objects via IPC
- Servers receive unforgeable client identities from the IPC mechanism
  - Servers can implement arbitrary access control policy
- No special mechanisms needed in the microkernel

Is this really true???

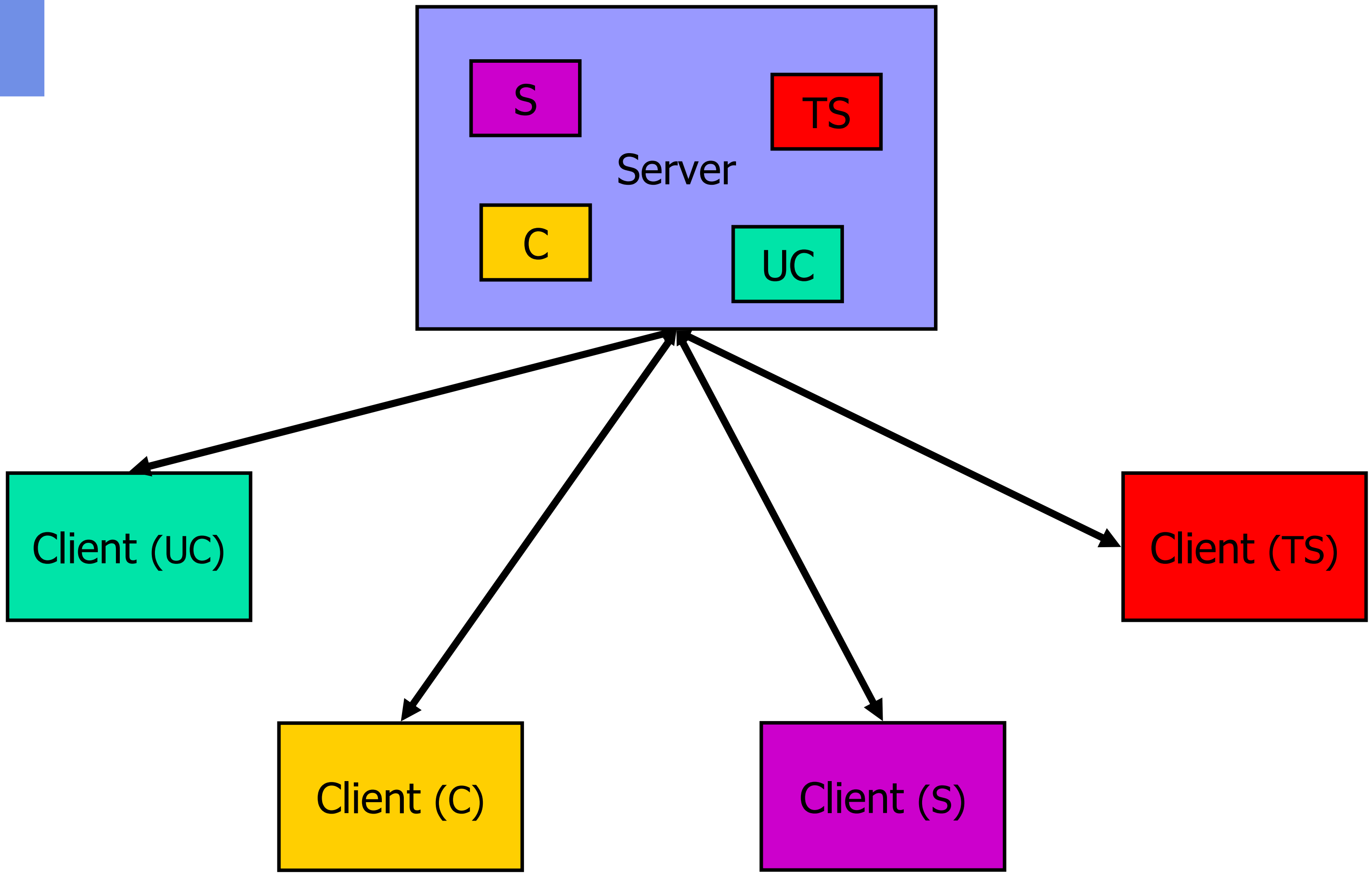
Example Policy

Multi Level Security (MLS) – Confidentiality

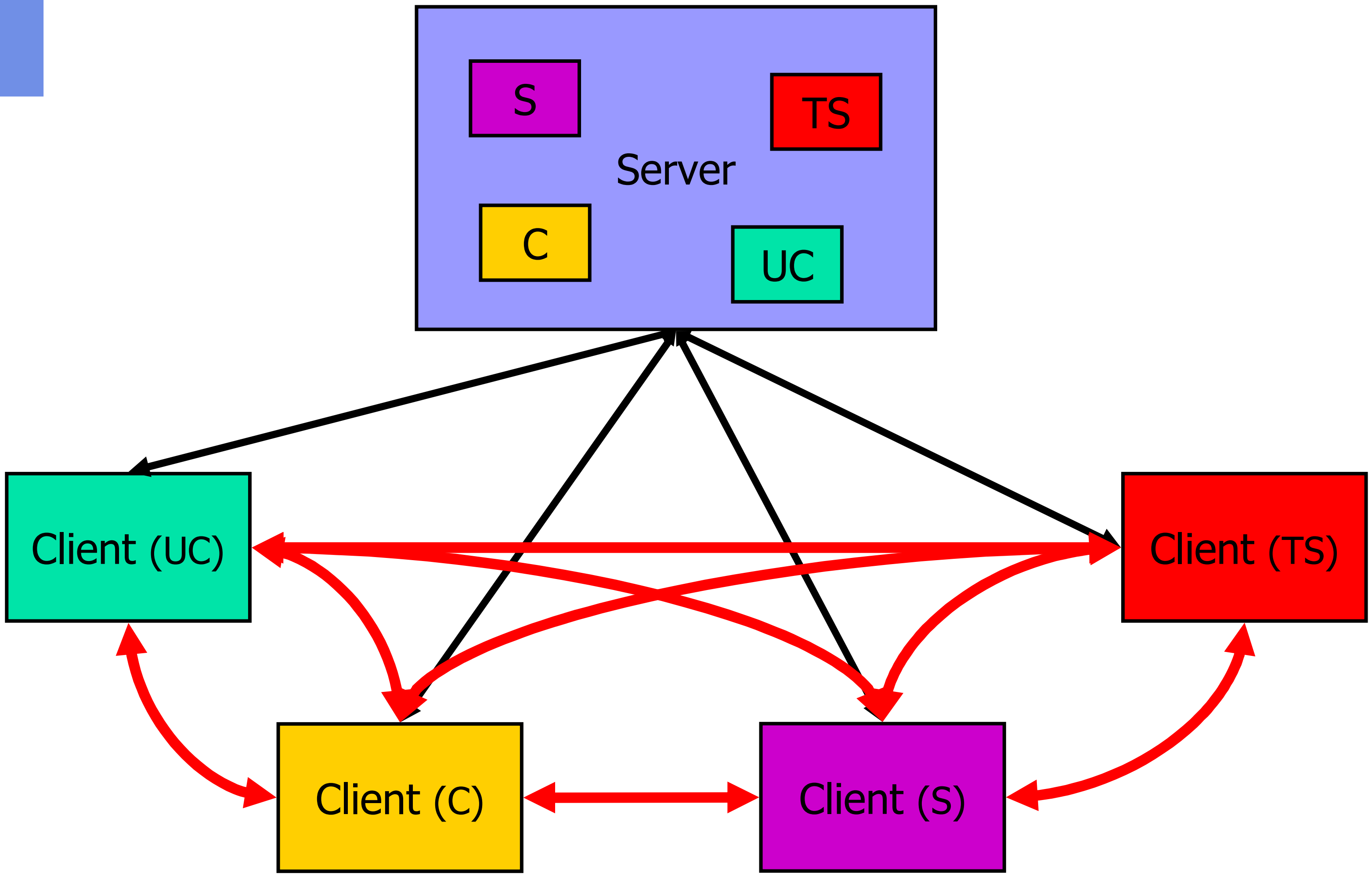
- Assign security levels to objects
  - Top Secret, Secret, Classified, Unclassified
    - TS > S > C > UC
- Assign security levels to subjects (users)
  - Top Secret, Secret, Classified, Unclassified

- Subject S can read object O iff
  - level (S) ≥ level (O)
- Subject S can write (append to) object O iff
  - level (S) ≤ level (O)

Example Policy
Multi Level Security (MLS) – Confidentiality



Problem

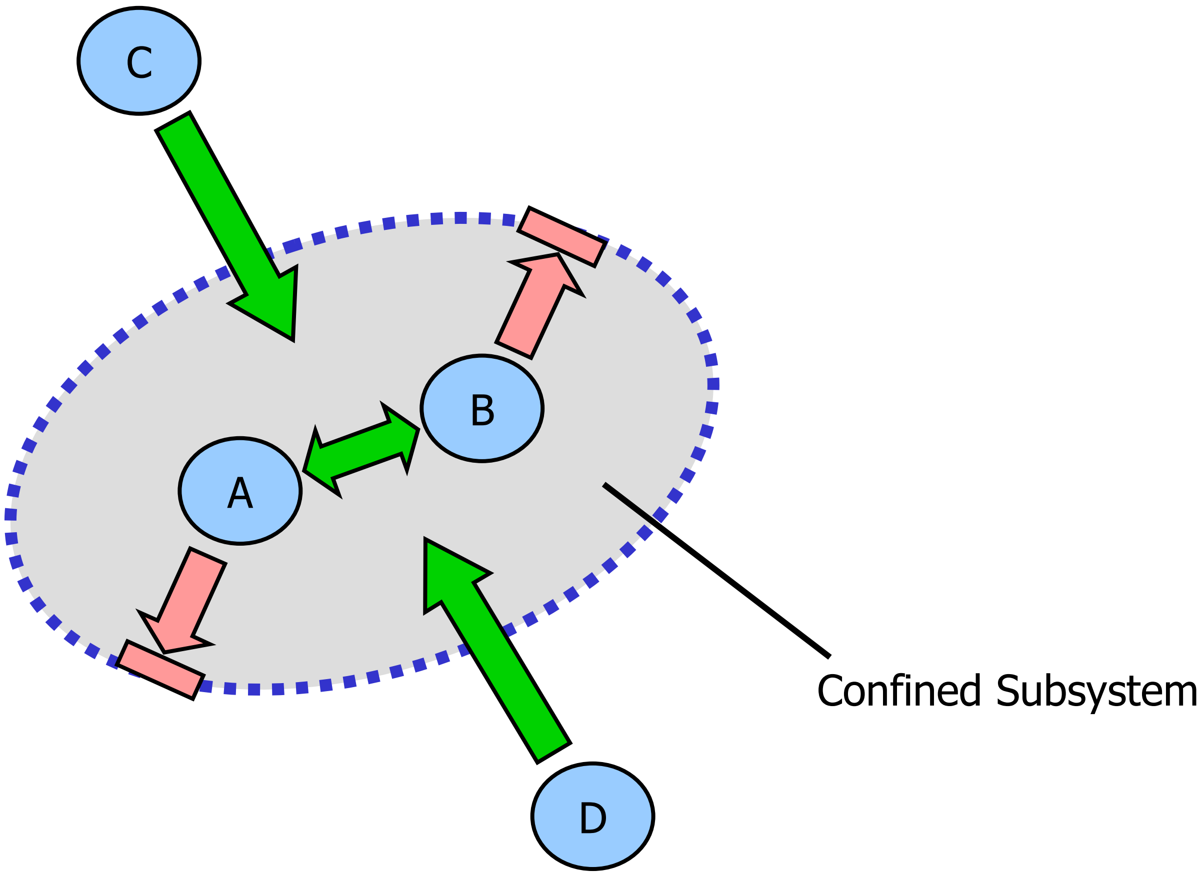


Conclusion

To control information flow we must control communication.

- We need mechanisms to not only implement a policy – we must also be able to enforce a policy
- Mechanism must be flexible enough to implement and enforce all relevant security policies

Confinement



Clans & Chiefs

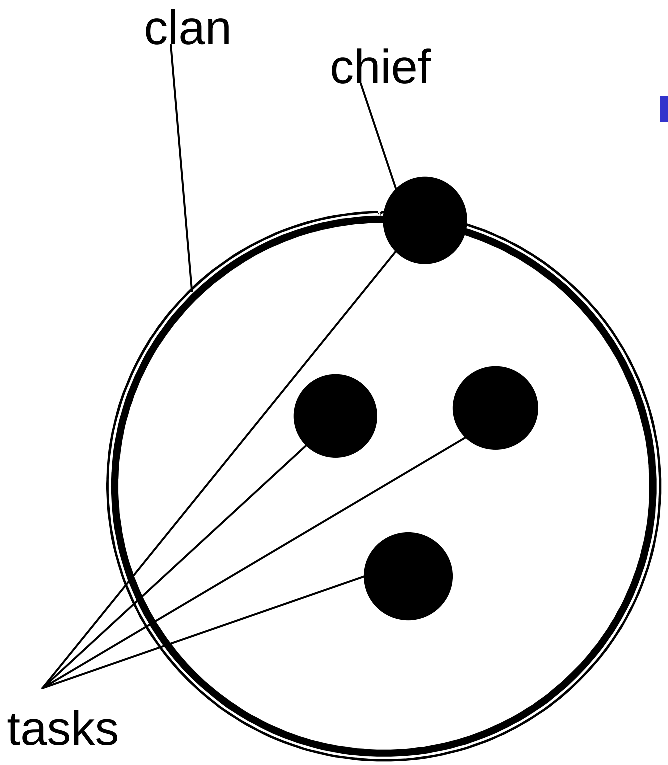
The Traditional L4 Approach

Clans & Chiefs

Within […] **systems based on direct message transfer, […] protection is essentially a matter of message control**. For the well known access control lists (acl) this can be done at the server level. But maintenance of large distributed acls becomes hard, when access rights change rapidly. […] To complement object (= passive entity) protection […], the kernel is able to restrict the outgoing message of a task (the subject). […]
A *clan* is a set of tasks headed by a *chief* task. Inside the clan all messages are transferred freely and the kernel guarantees message integrity. But whenever a message tries to cross a clan's borderline, regardless whether it is outgoing or incoming, it is redirected to the clan's chief. This chief may inspect the message (sender and receiver as well as contents) and decide whether or not it should be passed to the destination to which it was addressed. […]
Obviously subject restriction and local reference monitors can be implemented outside the kernel by means of clans. Since chief are tasks at user level, the clan concept allows more sophisticated and user definable checks as well as active control.

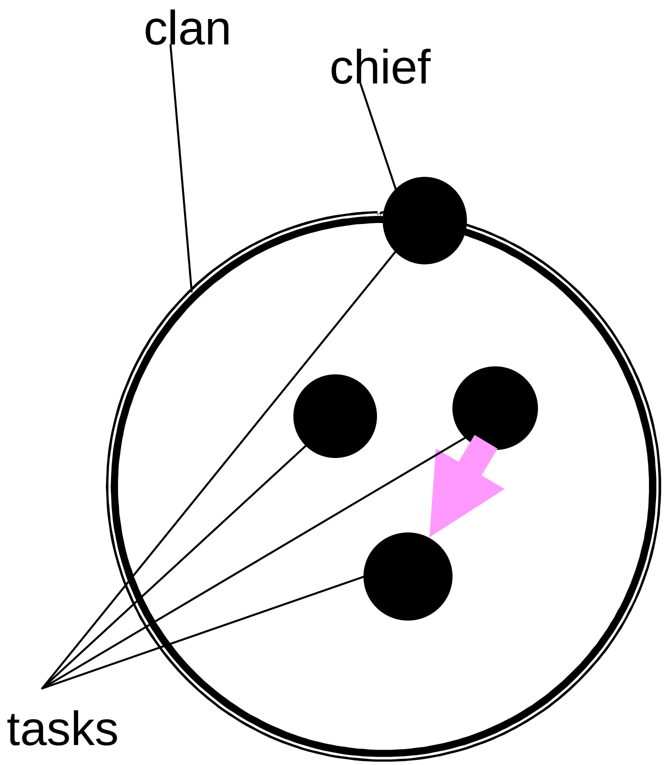
J. Liedtke: "Clans & Chiefs", In *12. GI/ITG-Fachtagung Architektur von Rechensystemen*, Springer Verlag, 1992, http://os.inf.tu-dresden.de/papers_ps/jochen/clansandchiefs.ps.gz

Clans & Chiefs



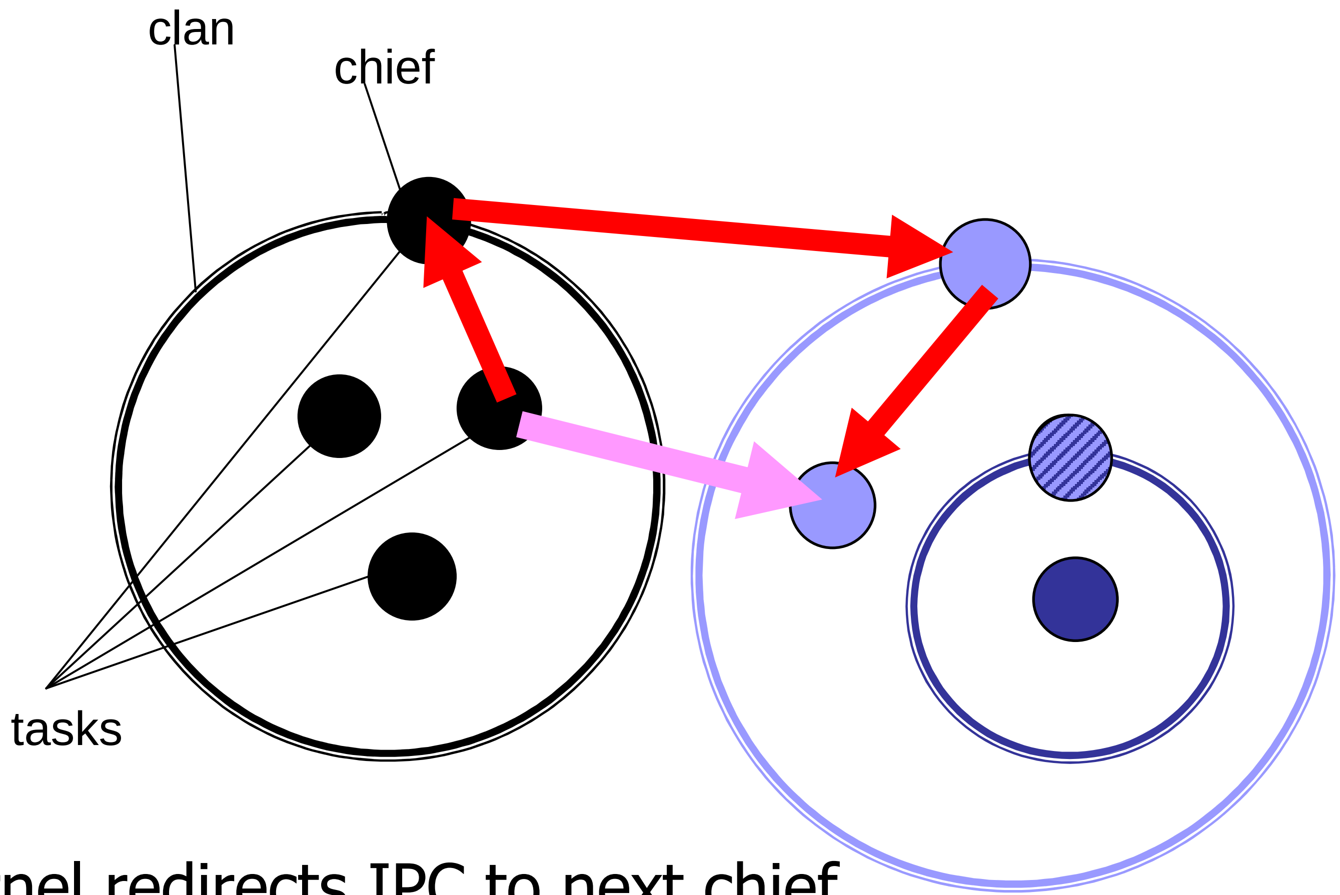
- A clan is a set of tasks headed by a chief task

Intra-Clan IPC



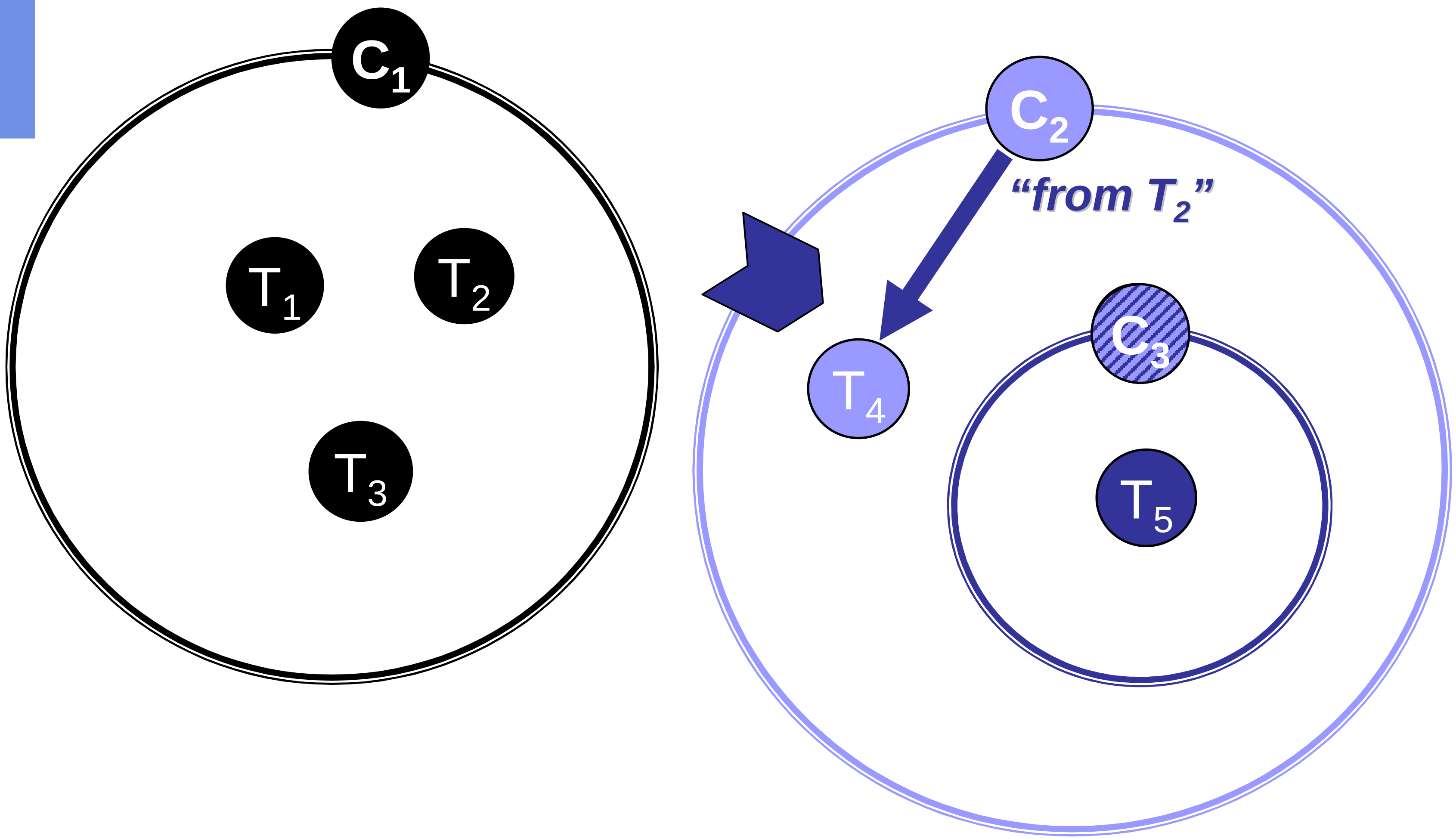
- Direct IPC by microkernel

Inter-Clan IPC

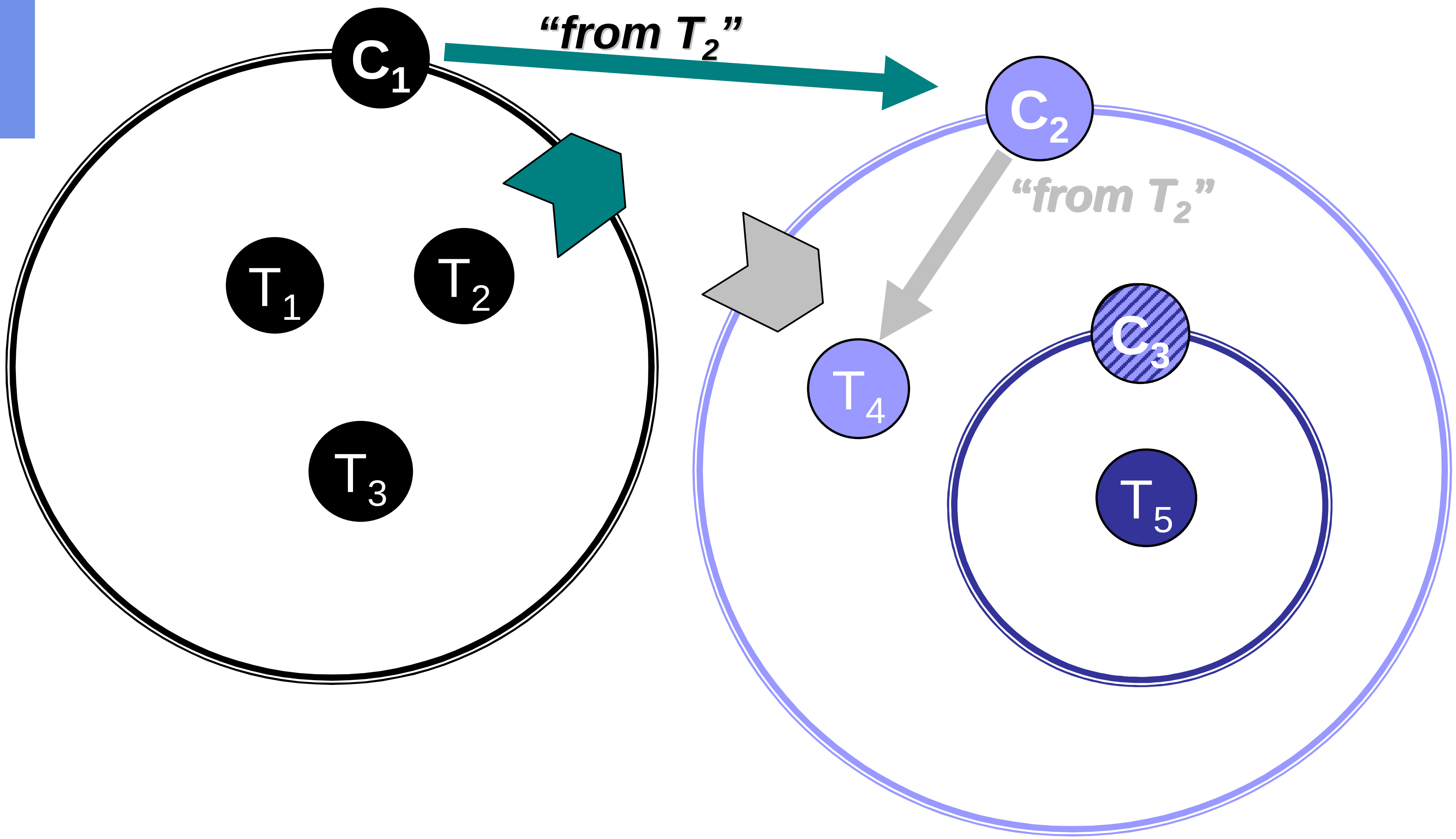


- Microkernel redirects IPC to next chief
- Chief (user task) can forward IPC or modify or …

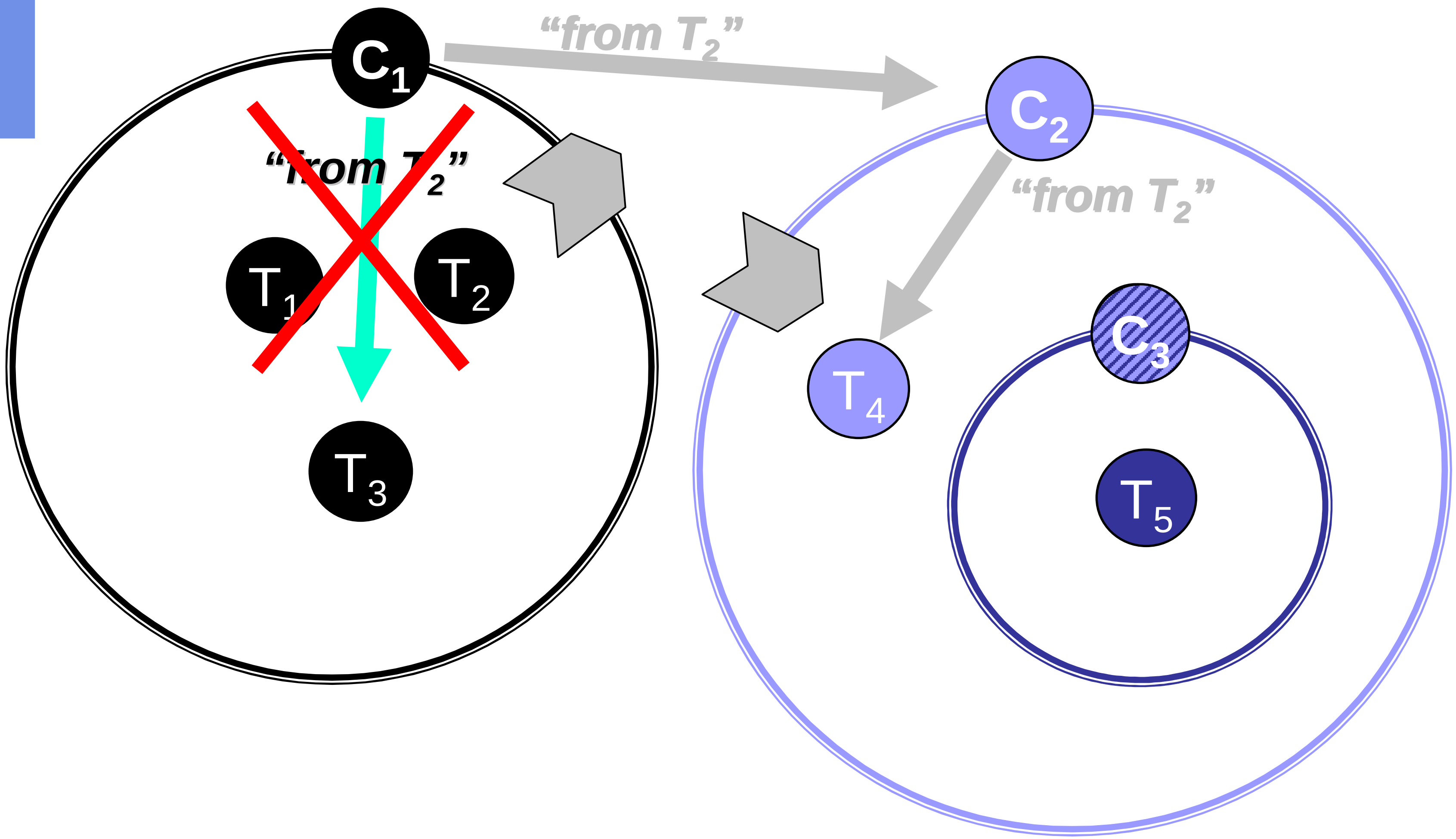
Direction-Preserving Deceiving



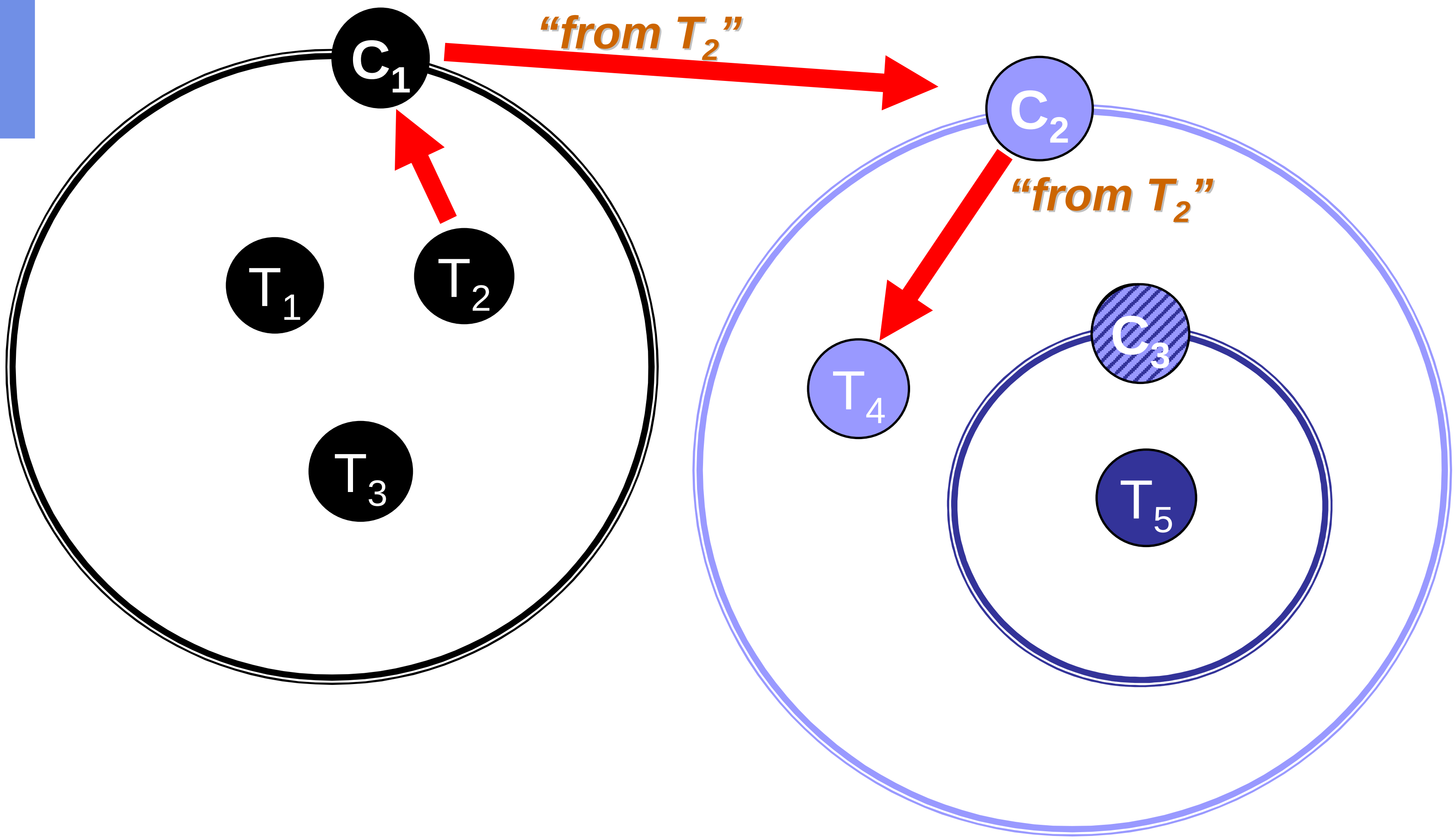
Direction-Preserving Deceiving



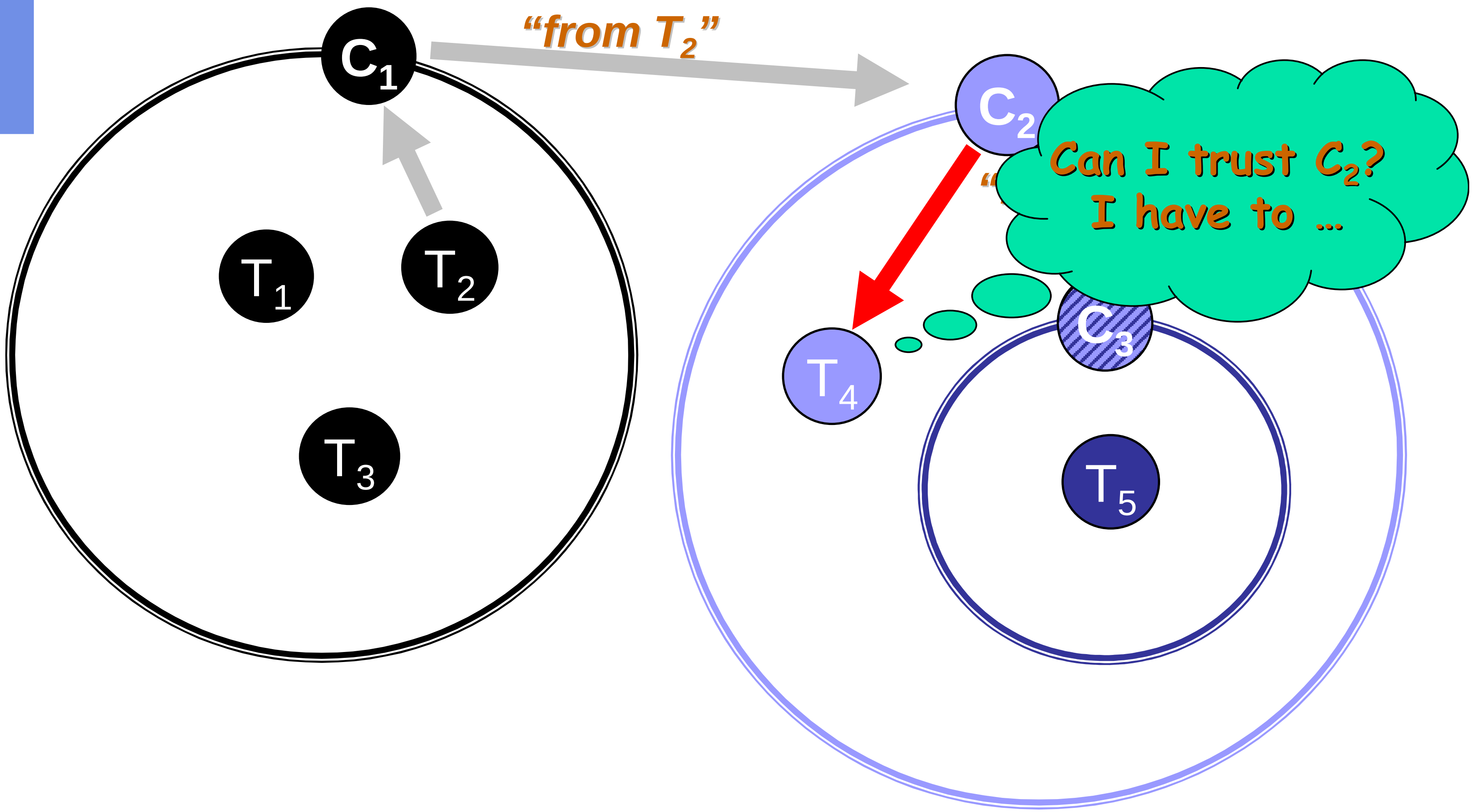
Direction-Preserving Deceiving



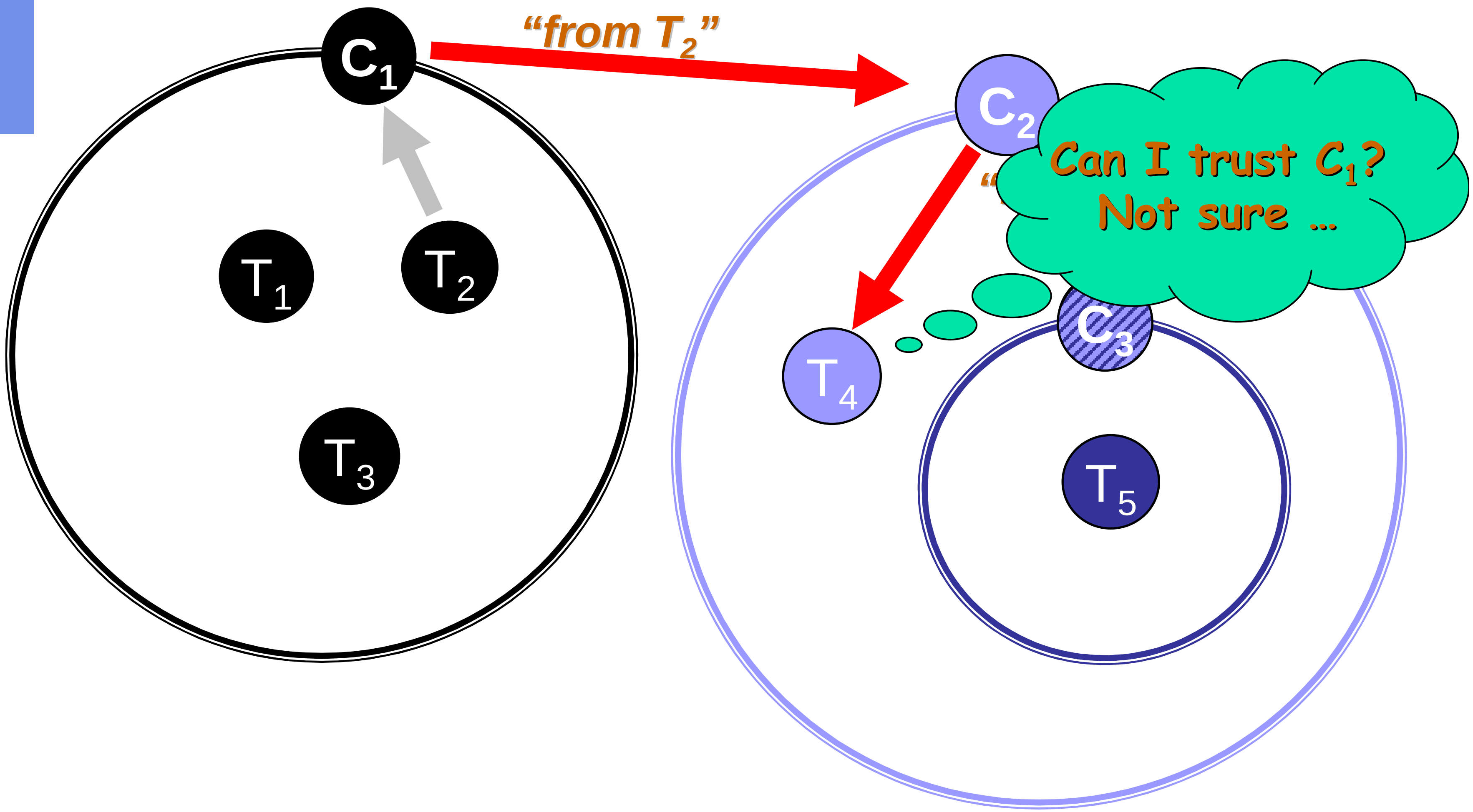
Direction-Preserving Deceiving



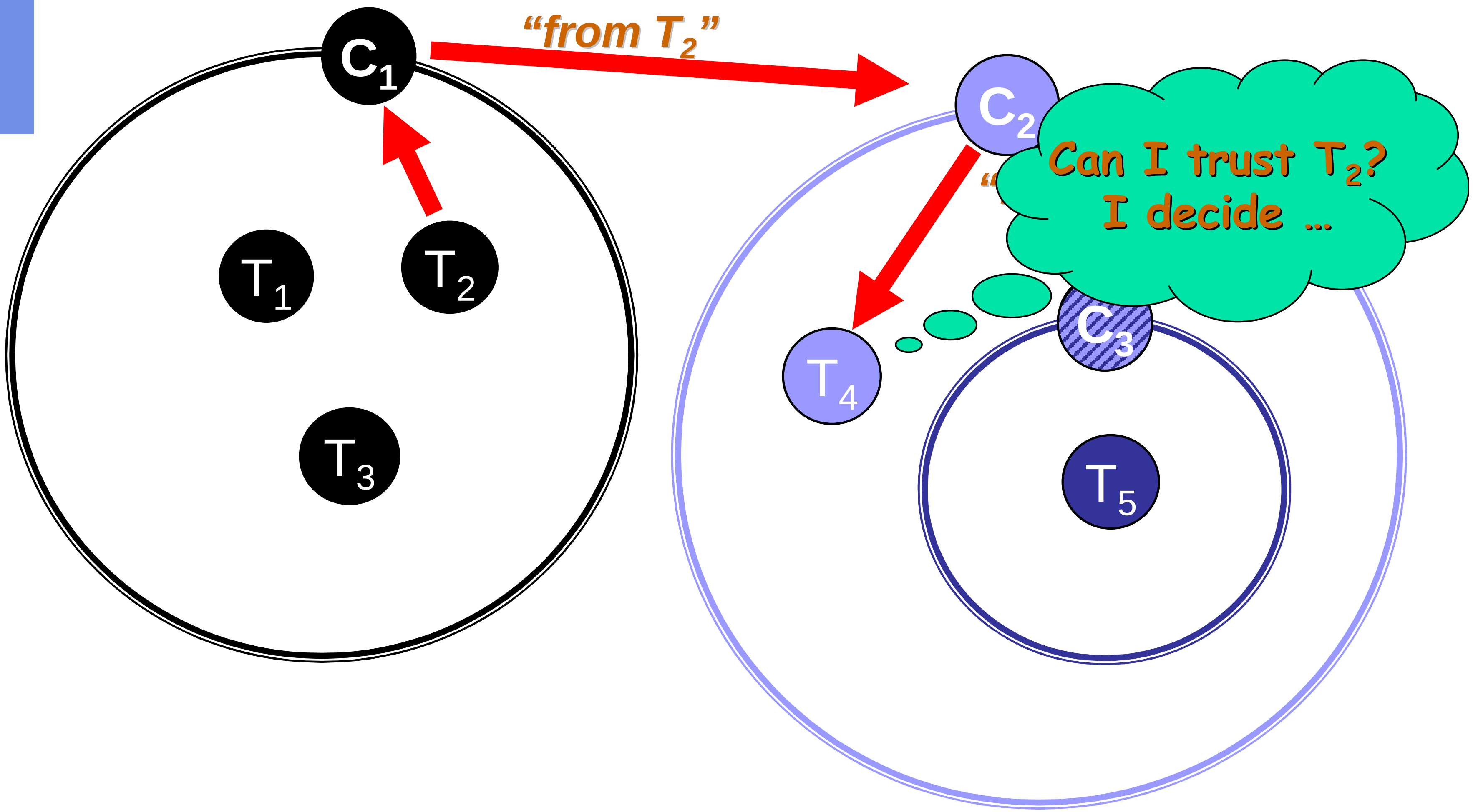
Direction-Preserving Deceiving



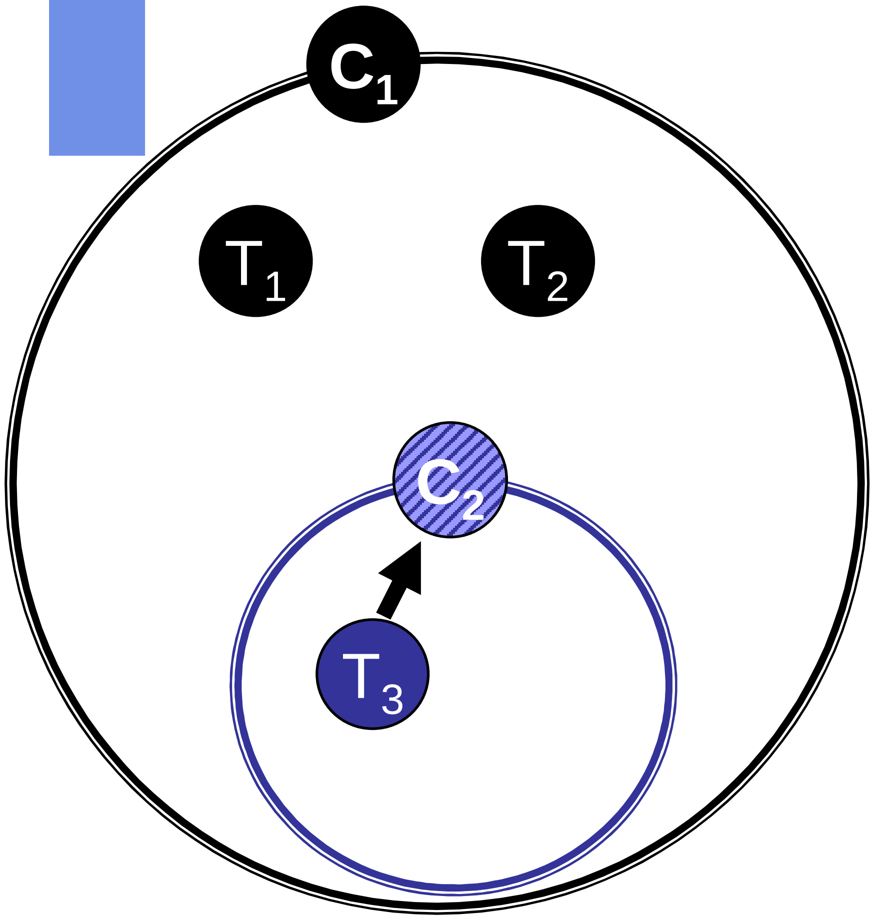
Direction-Preserving Deceiving



Direction-Preserving Deceiving

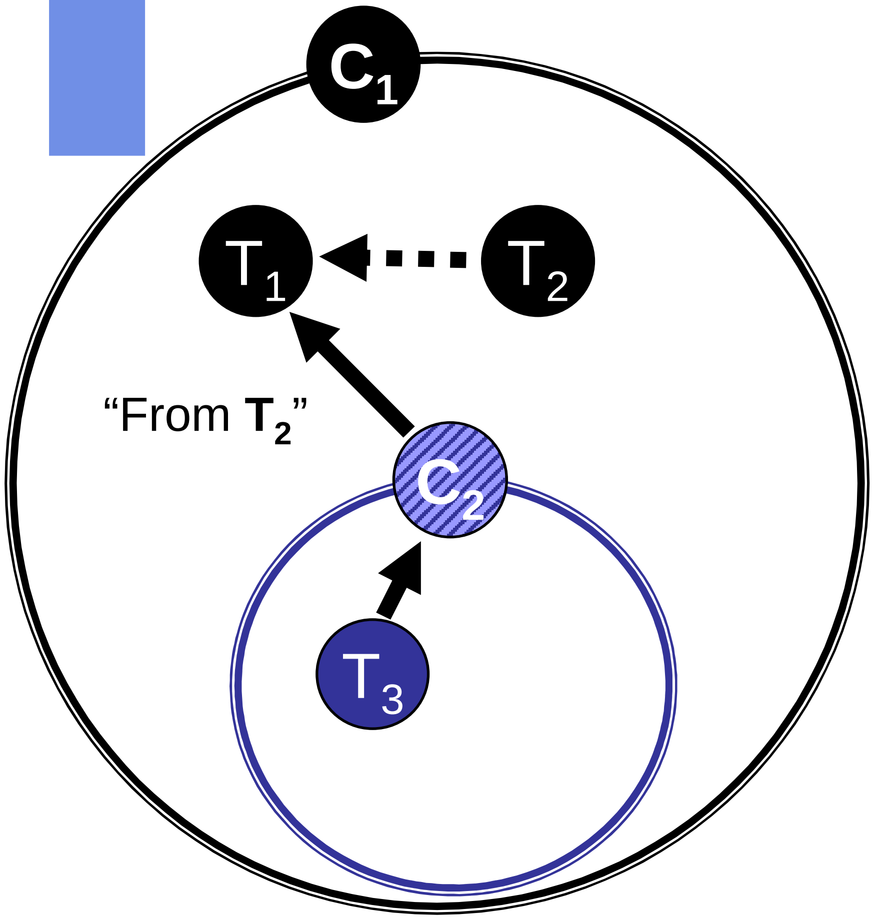


Example



Direction-Preserving-Deceiving (DPD) is a simple mechanism to realize security.
Imagine the blue task is a tool you have from the Internet. The blue thread $\mathbf{T_3}$ wants to make $\mathbf{T_1}$ replace some private data from $\mathbf{T_1}$ (e.g. ARP cache entry for $\mathbf{T_2}$).

Example

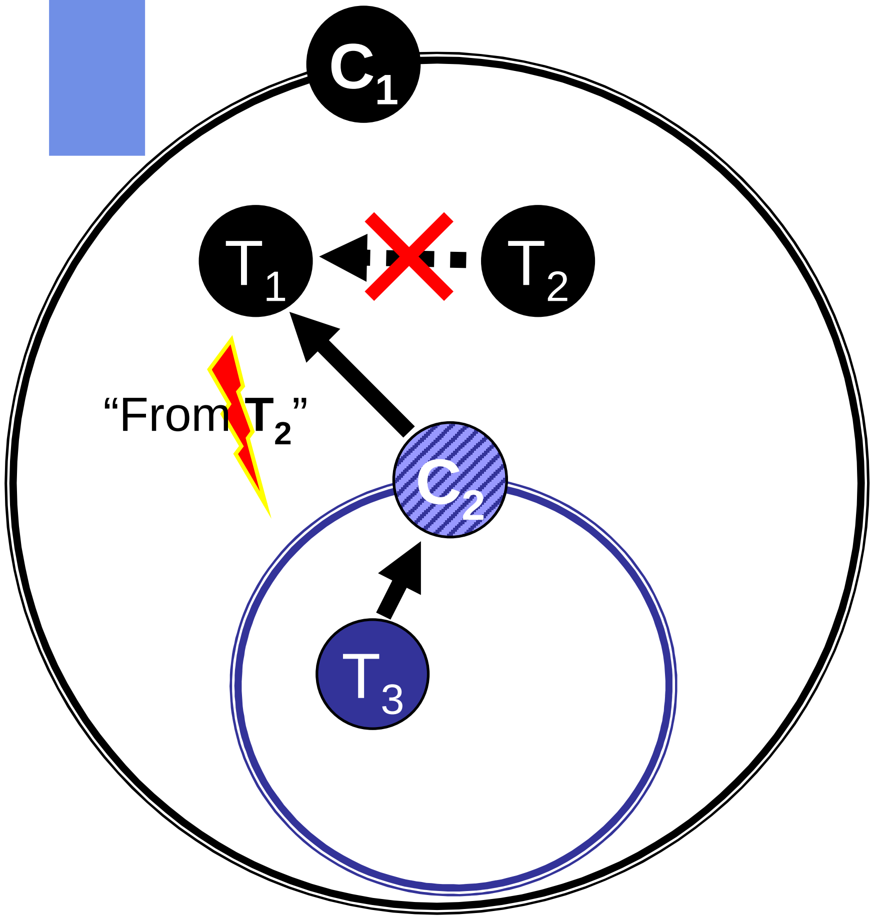


Direction-Preserving-Deceiving (DPD) is a simple mechanism to realize security.

Imagine the blue task is a tool you have from the Internet. The blue thread $\mathbf{T_3}$ wants to make $\mathbf{T_1}$ replace some private data from $\mathbf{T_1}$ (e.g. ARP cache entry for $\mathbf{T_2}$).

Without DPD there is no relevant security: The chief $\mathbf{C_2}$ *could* send an IPC to $\mathbf{T_1}$ pretending that it came from $\mathbf{T_2}$.

Example



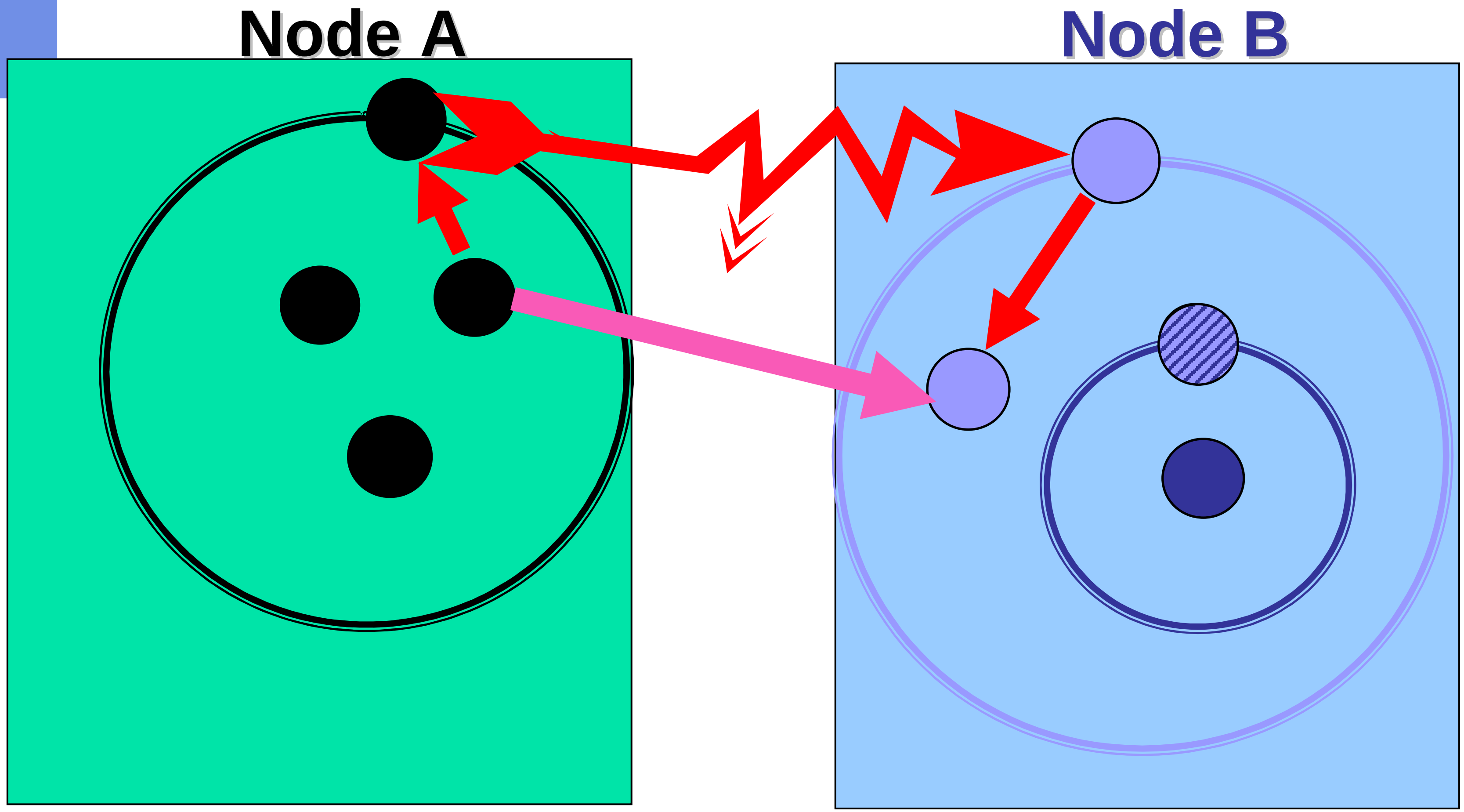
Direction-Preserving-Deceiving (DPD) is a simple mechanism to realize security.

Imagine the blue task is a tool you have from the Internet. The blue thread $\mathbf{T_3}$ wants to make $\mathbf{T_1}$ replace some private data from $\mathbf{T_1}$ (e.g. ARP cache entry for $\mathbf{T_2}$).

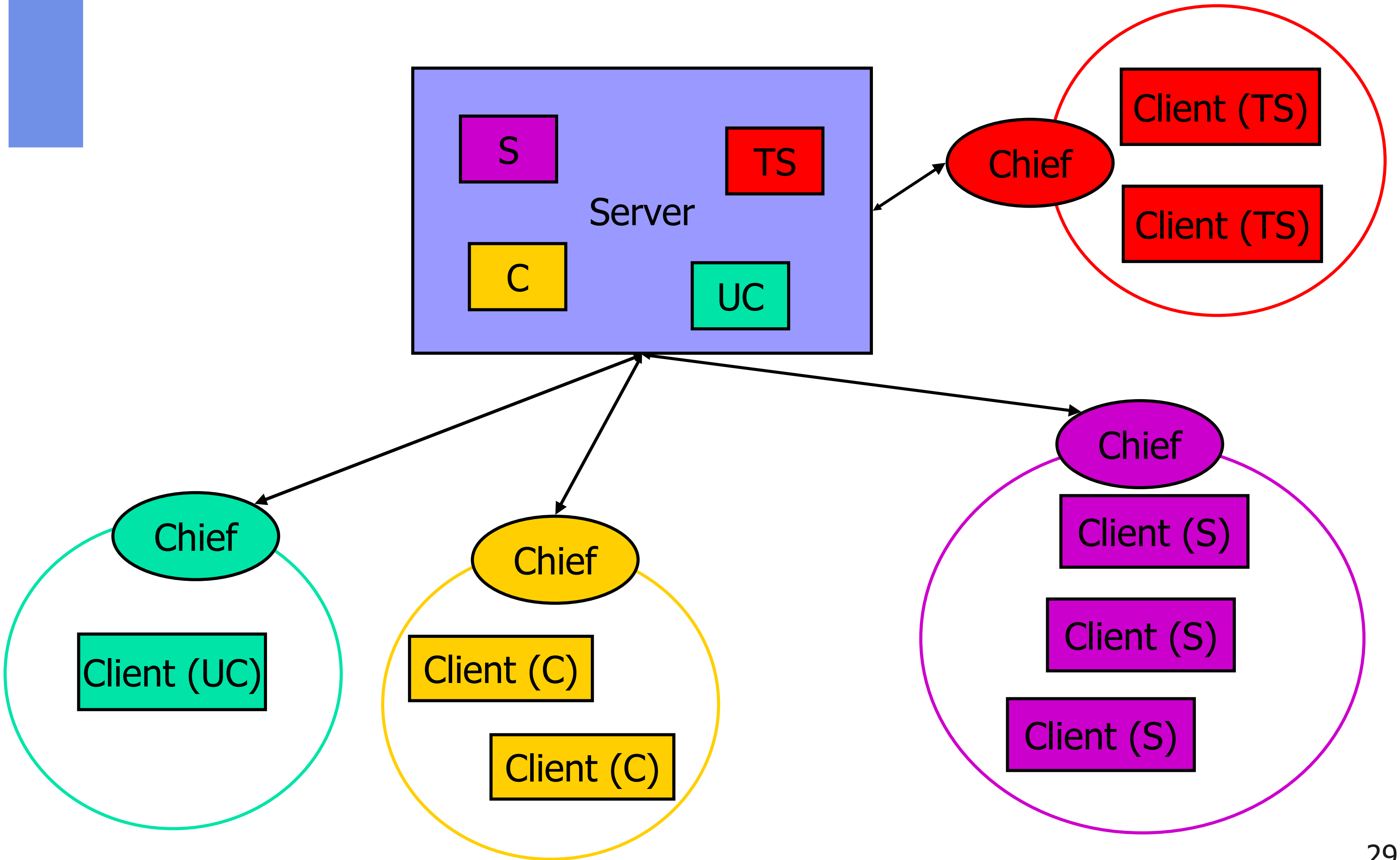
Without DPD there is no relevant security: The chief $\mathbf{C_2}$ *could* send an IPC to $\mathbf{T_1}$ pretending that it came from $\mathbf{T_2}$.

The important fact is that **with** DPD, $\mathbf{T_1}$ definitely knows that all messages it gets from $\mathbf{C_2}$ came from inside the clan $\mathbf{C_2}$. Vice versa is the same.

Remote IPC



Secure System Using Clans & Chiefs

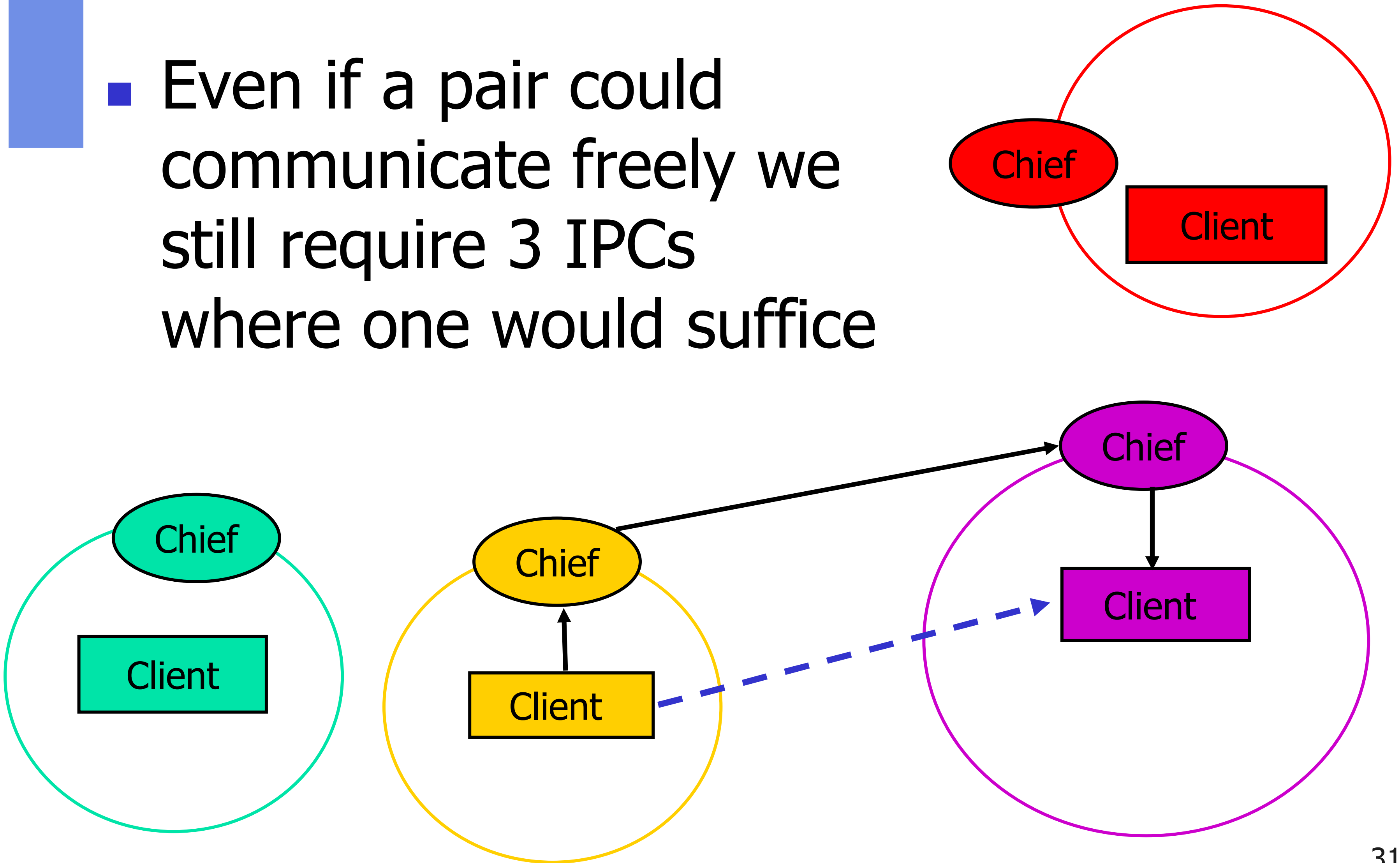


Problems with Clans & Chiefs

- Static
  - A chief is assigned when task is started
    - If we might want to control IPC, we must always assign a chief
- General case requires many more IPCs
  - Every task has its own chief

The Most General System Configuration

- Even if a pair could communicate freely we still require 3 IPCs where one would suffice

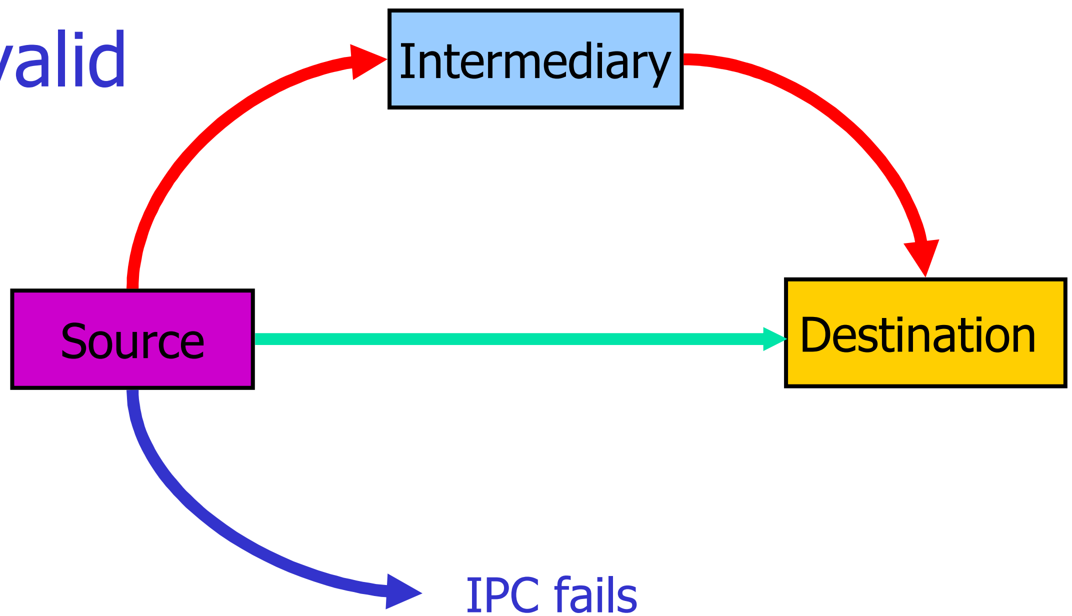


Generic IPC Redirection

Flexibility and Dynamic Reconfiguration

IPC Redirection

- For each source and destination we actually deliver to X, where X is one of
  - Destination
  - Intermediary
  - Invalid

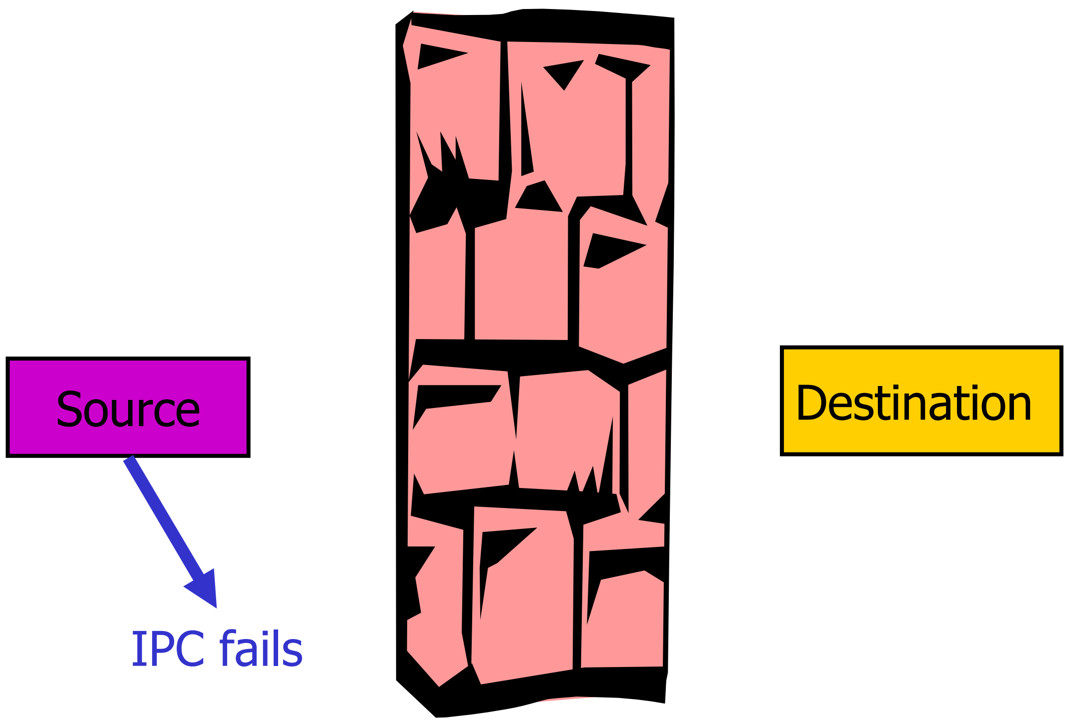


IPC Redirection

- If X = Destination
  - We have a fast path when source and destination can communicate freely

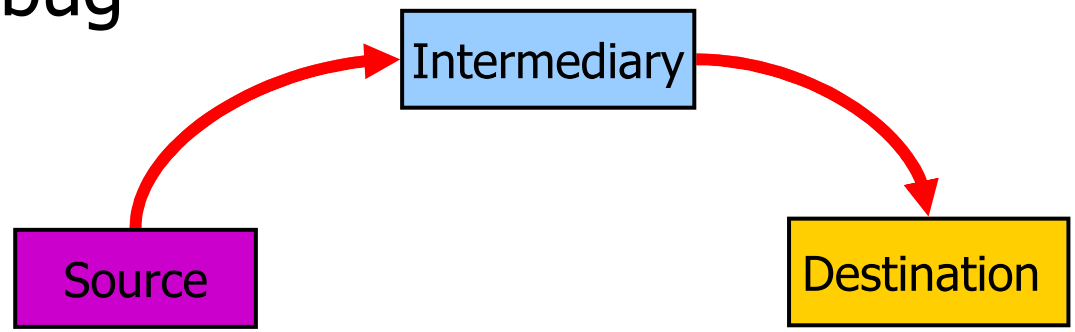
IPC Redirection

- If X = Invalid
  - We have a barrier that prevents communication completely



IPC Redirection

- If X = Intermediary
  - Enforce security policy
    - Monitor, analyze, reject, modify each IPC
  - Audit communication
  - Debug

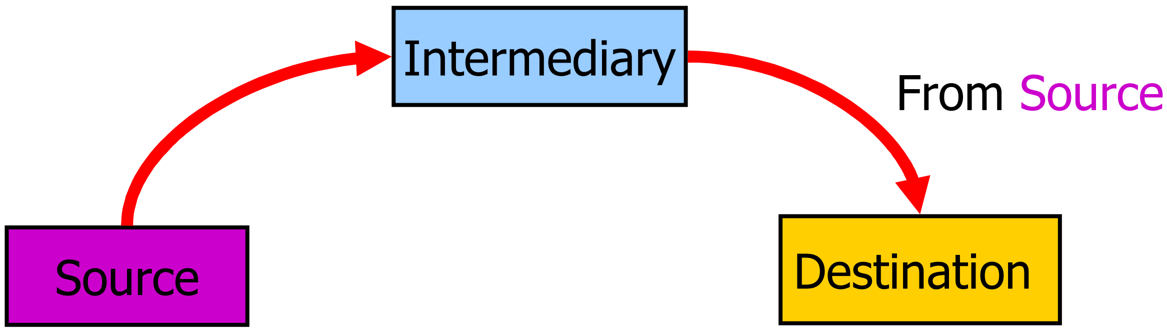


Deception

- Intermediaries must be able to deceive the destination into believing the intermediary is the original source
- An intermediary (I) can impersonate a source (S) in IPC to a destination (D)
  - I [S] ➡ D
    - If Redirection (S, D) = I, or
    - Redirection (S, D) = X and I [X] ➡ D (recursive)

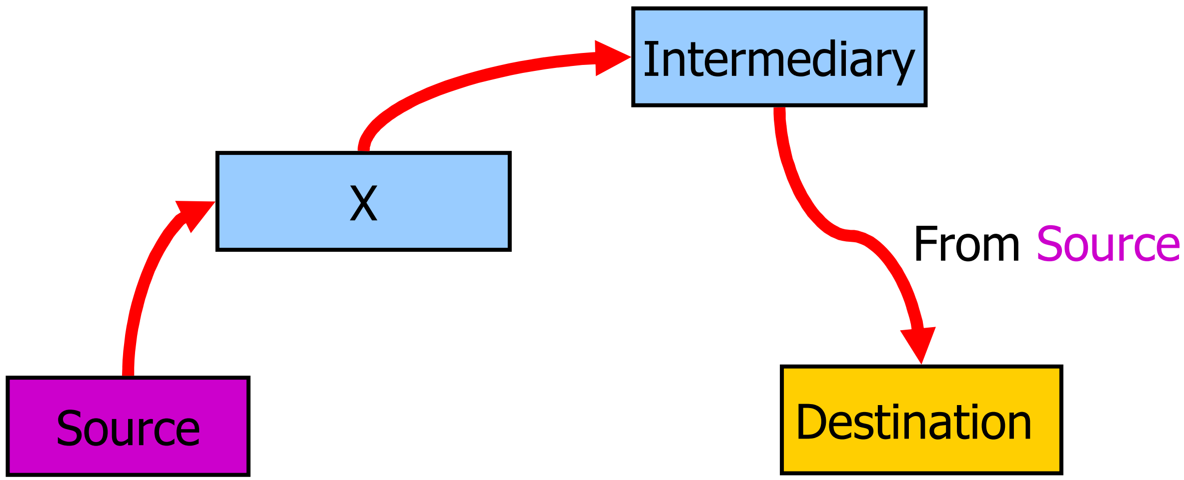
Deception
Case 1

- I [S] ➡ D if Redirection (S,D) = I

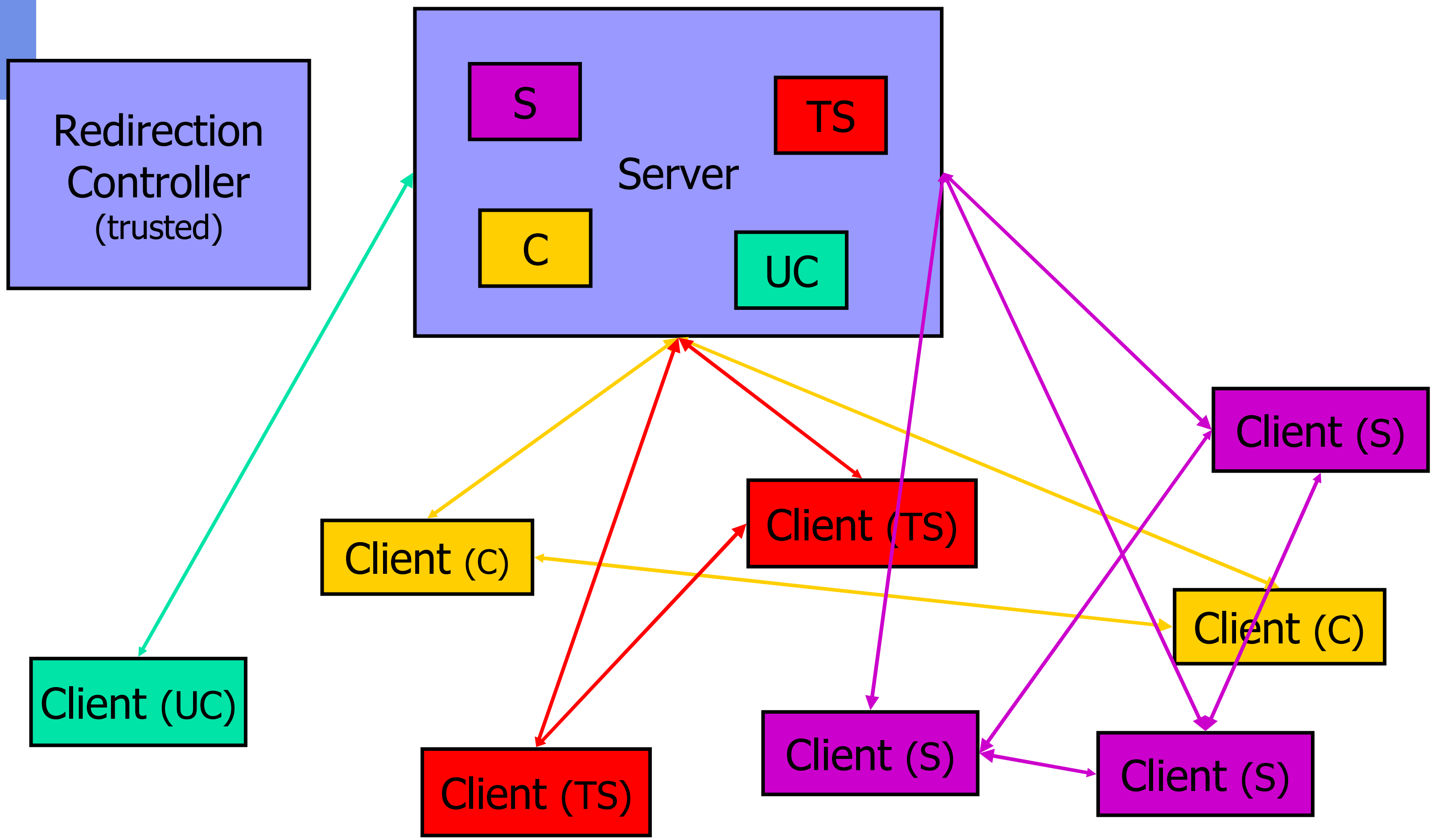


Deception
Case 2

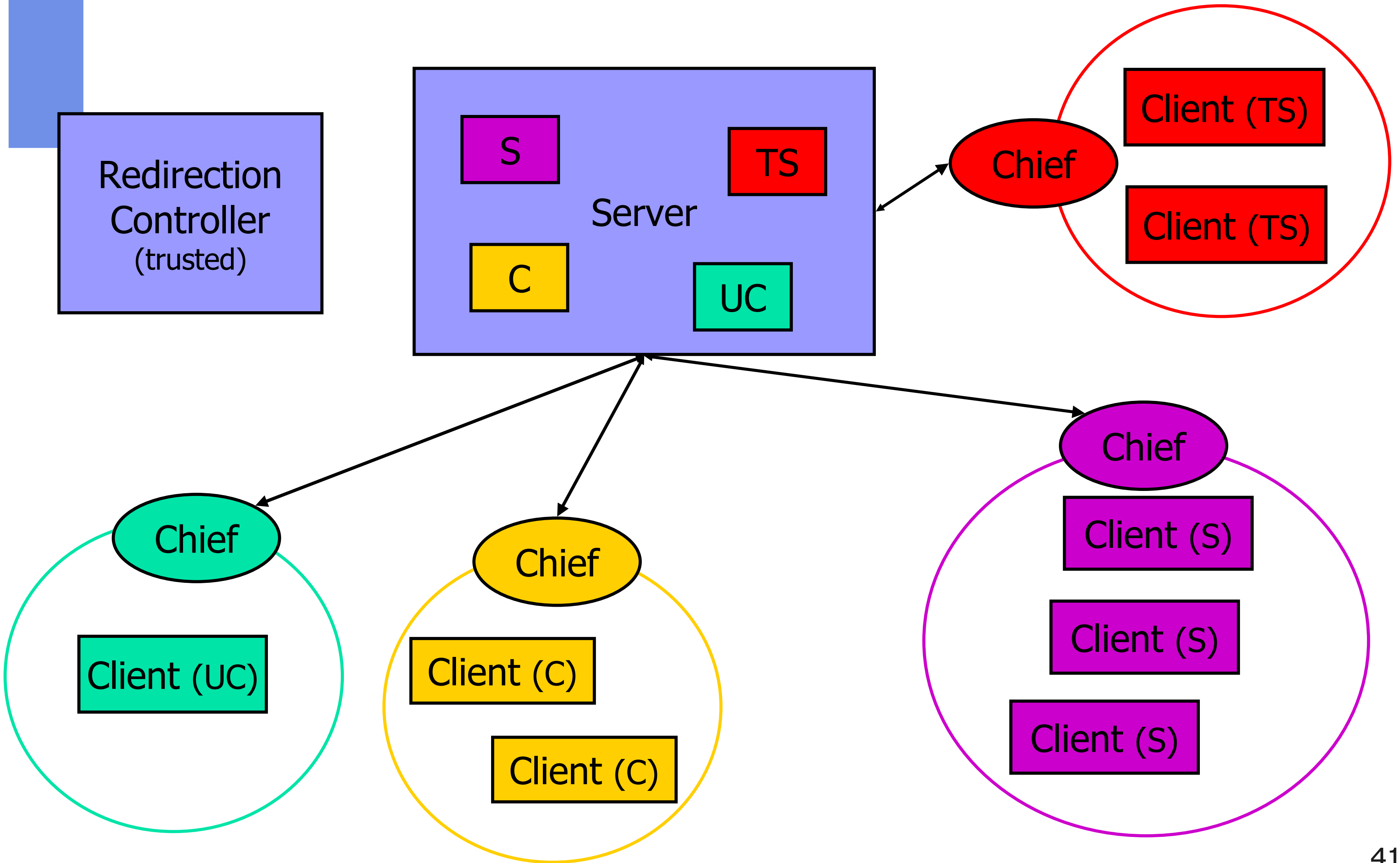
- I [S] ➡ D if Redirection (S,D) = X, and I [X] ➡ D (recursive)



Secure System Using IPC Redirection

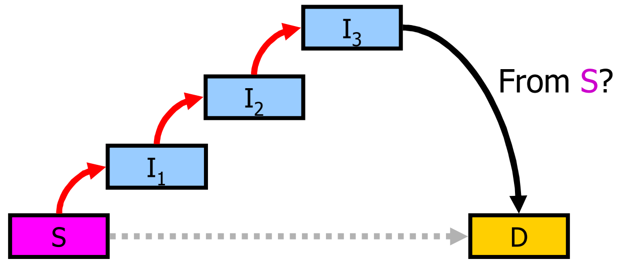


Clans & Chiefs Using IPC Redirection

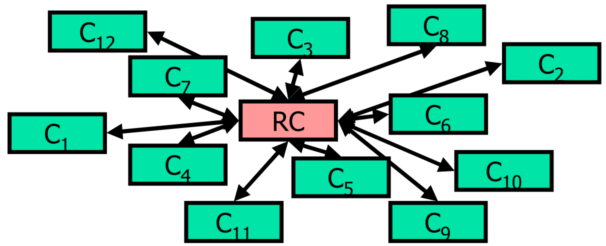


General IPC Redirection
Issues

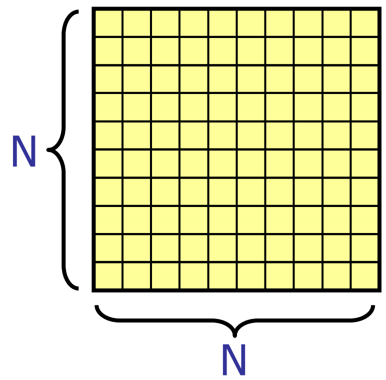
- Recursive operation
  - Can be expensive



- Centralized controller
  - Possible bottleneck



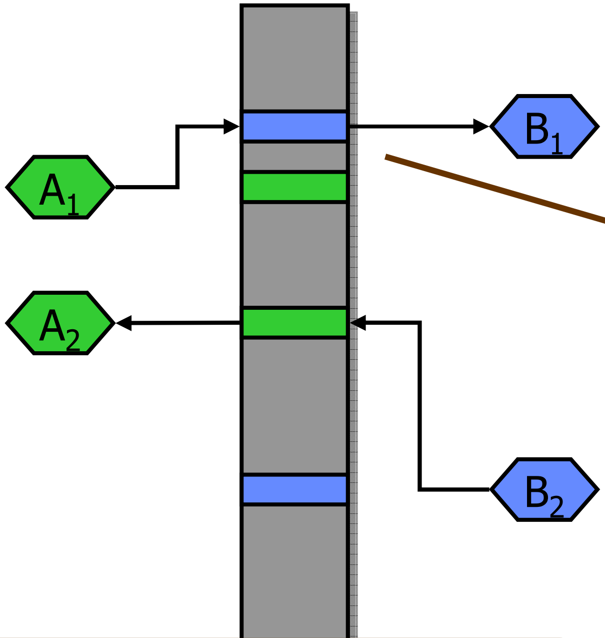
- Massive redirection structures
  - N×N array (N = num. threads)



Virtual Communication Channels

Decentralized IPC Management

Communication Spaces
Current Model: Single Global Space

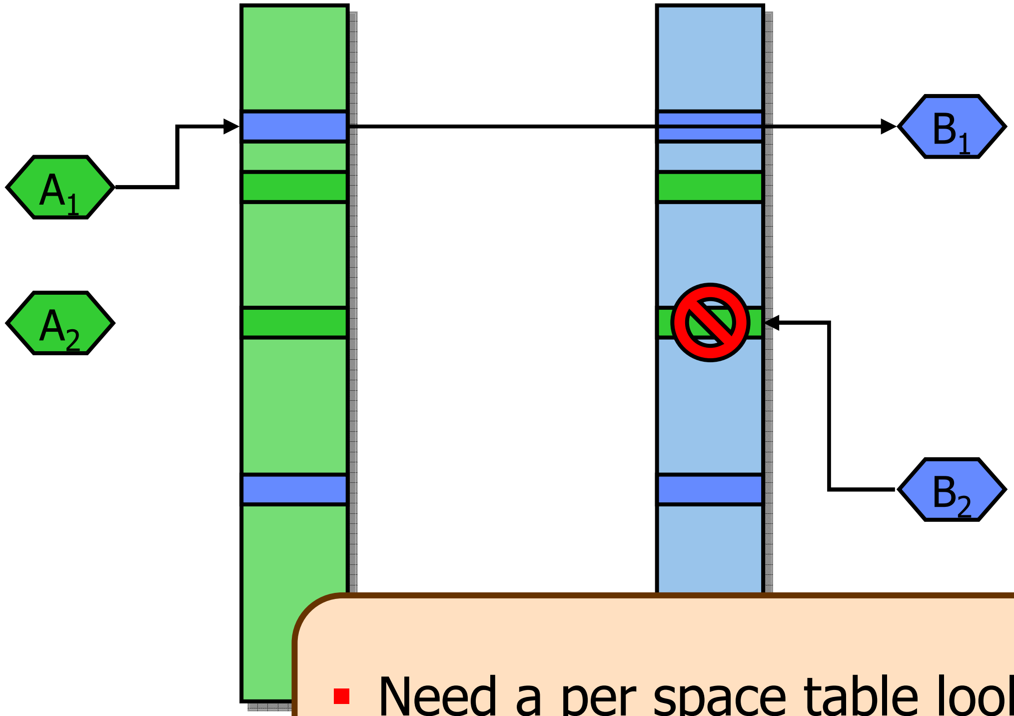


- Direct naming using global thread ID
- Extremely fast (no indirection)

Must be able to restrict access rights on a per address space basis

Communication Spaces

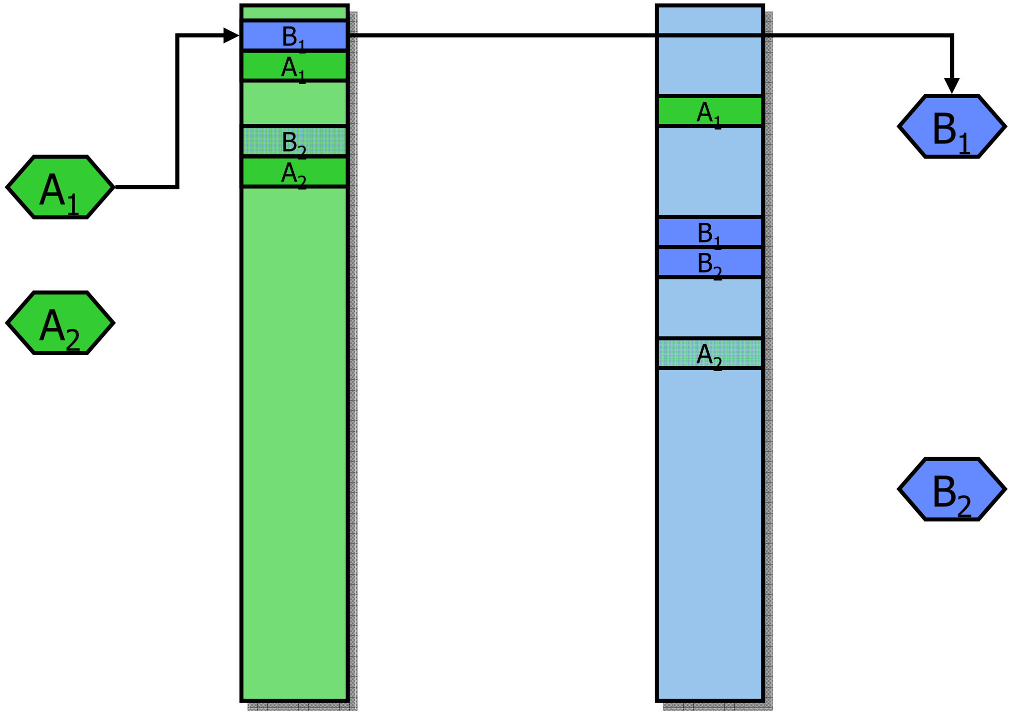
Possible Solution: Per Space Access Rights



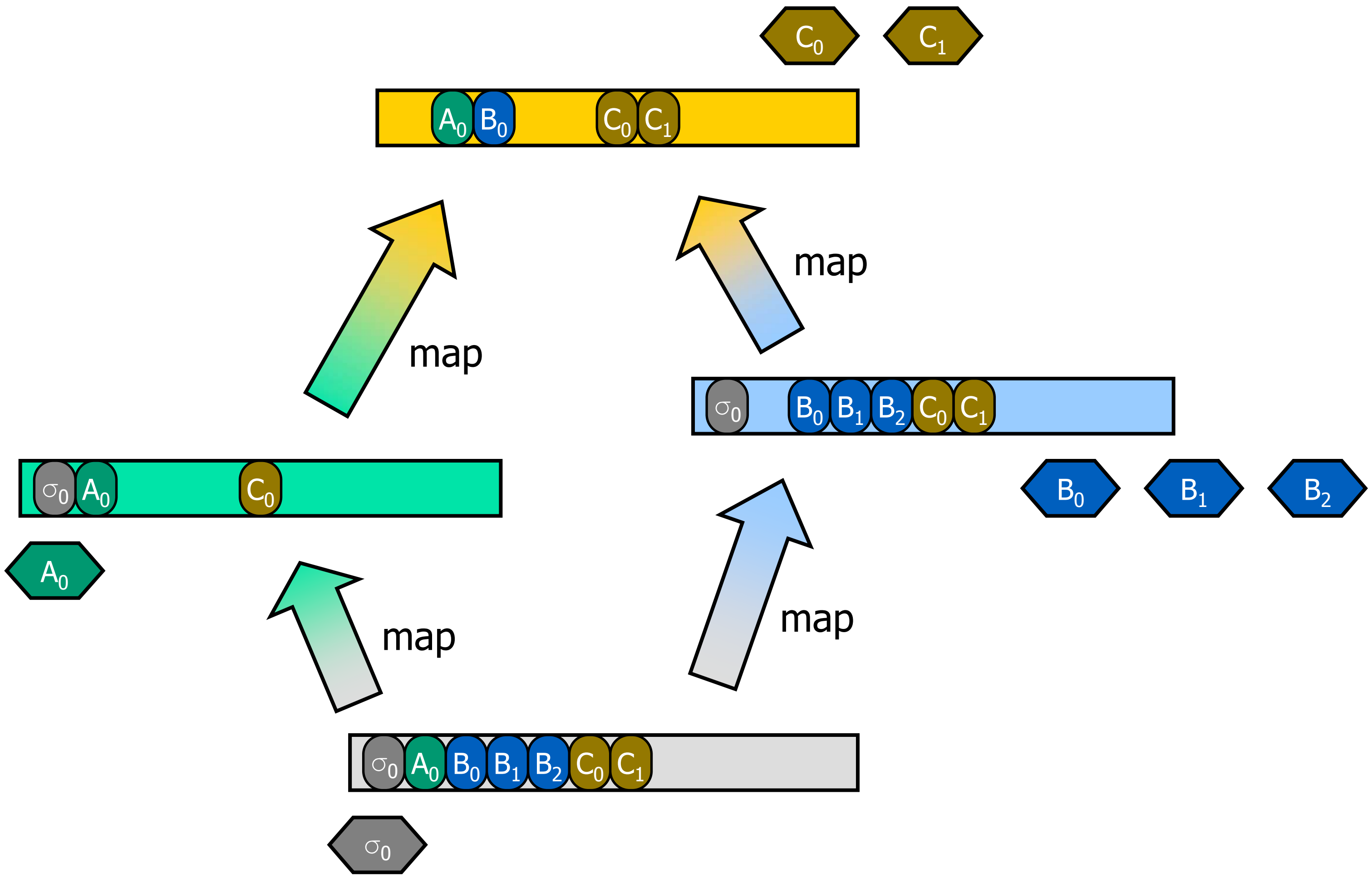
- Need a per space table lookup
- Might as well add indirection

Communication Spaces

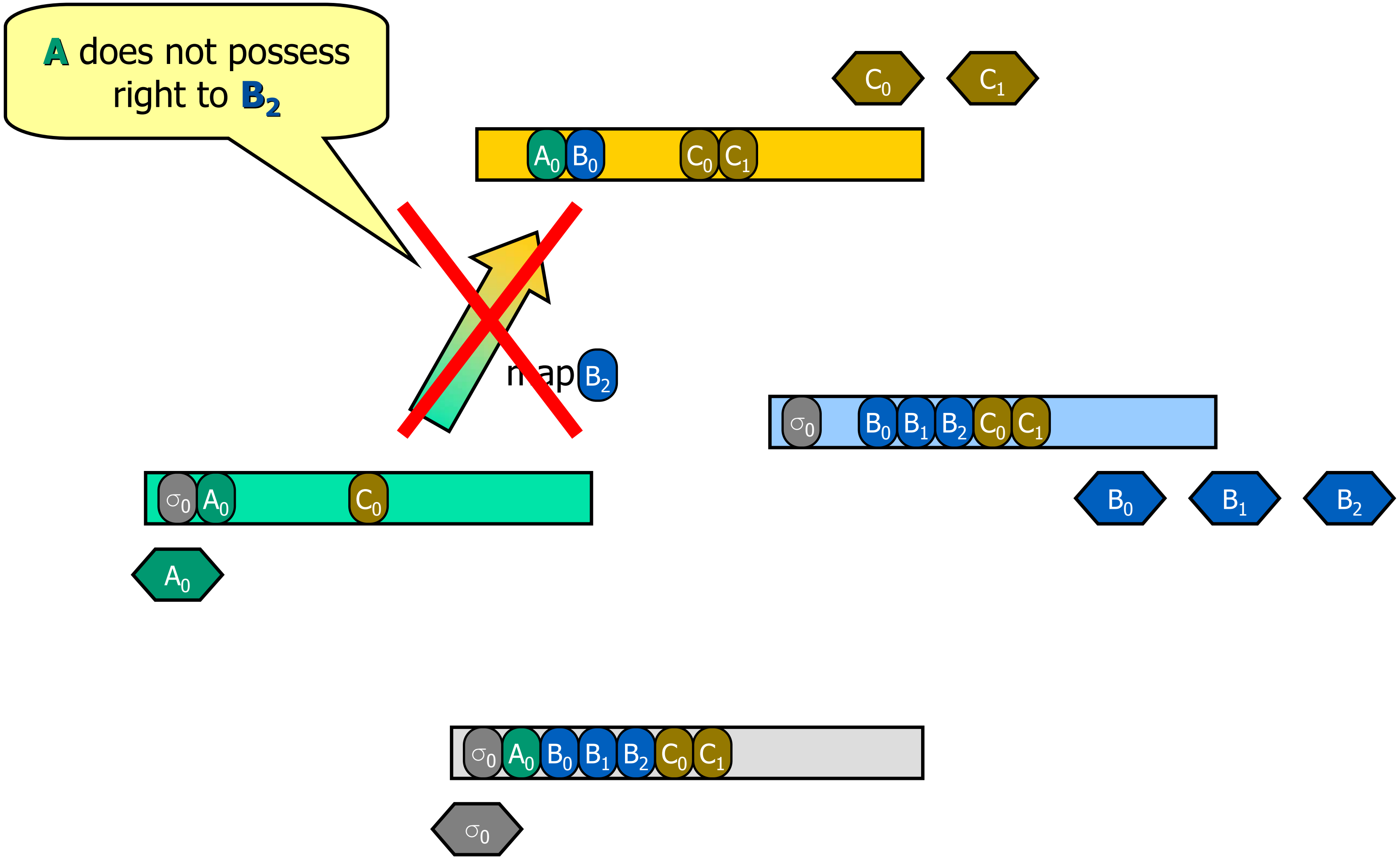
Better Solution: Per Space Indirection Table



Communication Spaces

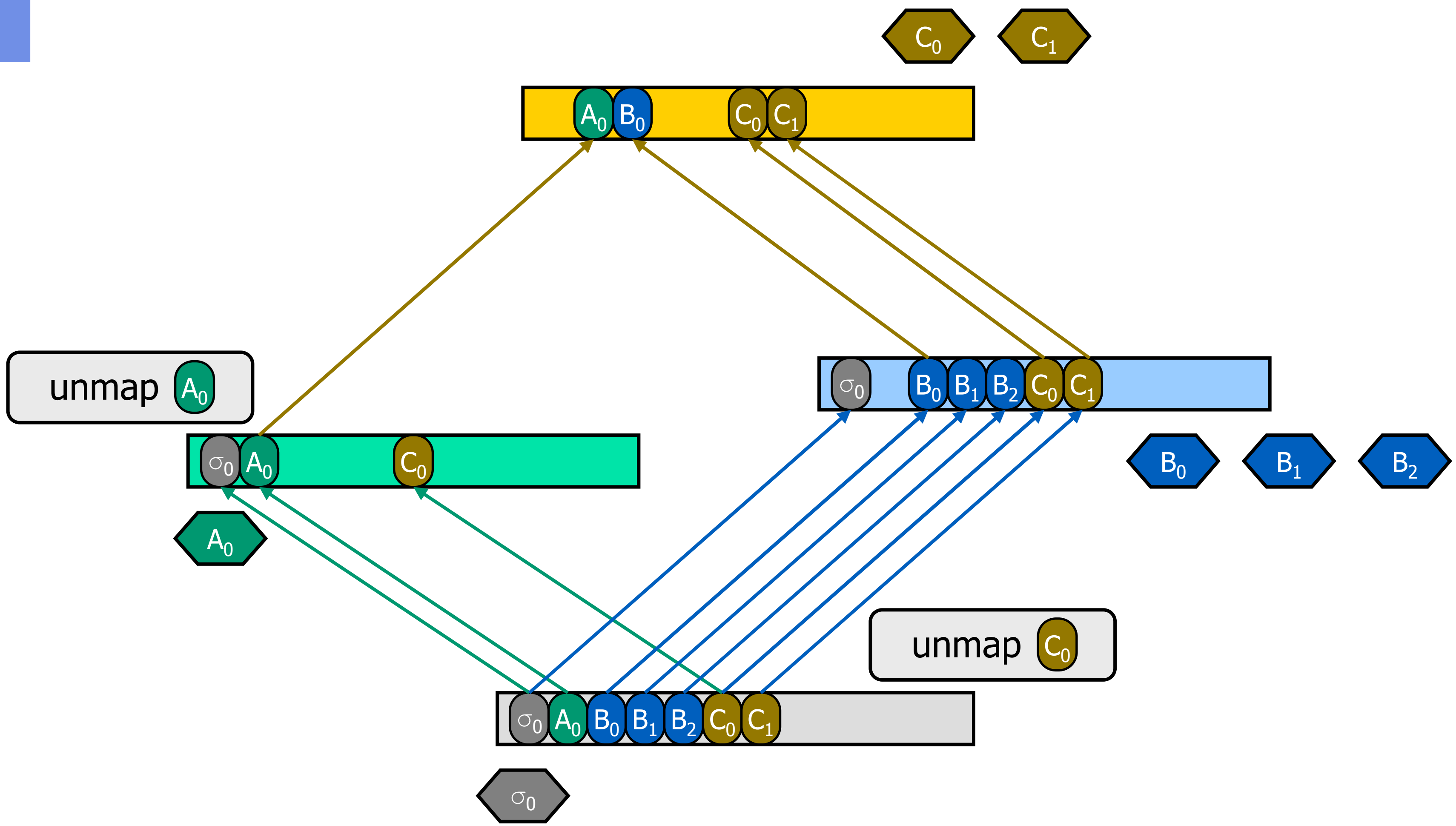


Communication Spaces
Mapping Access Rights

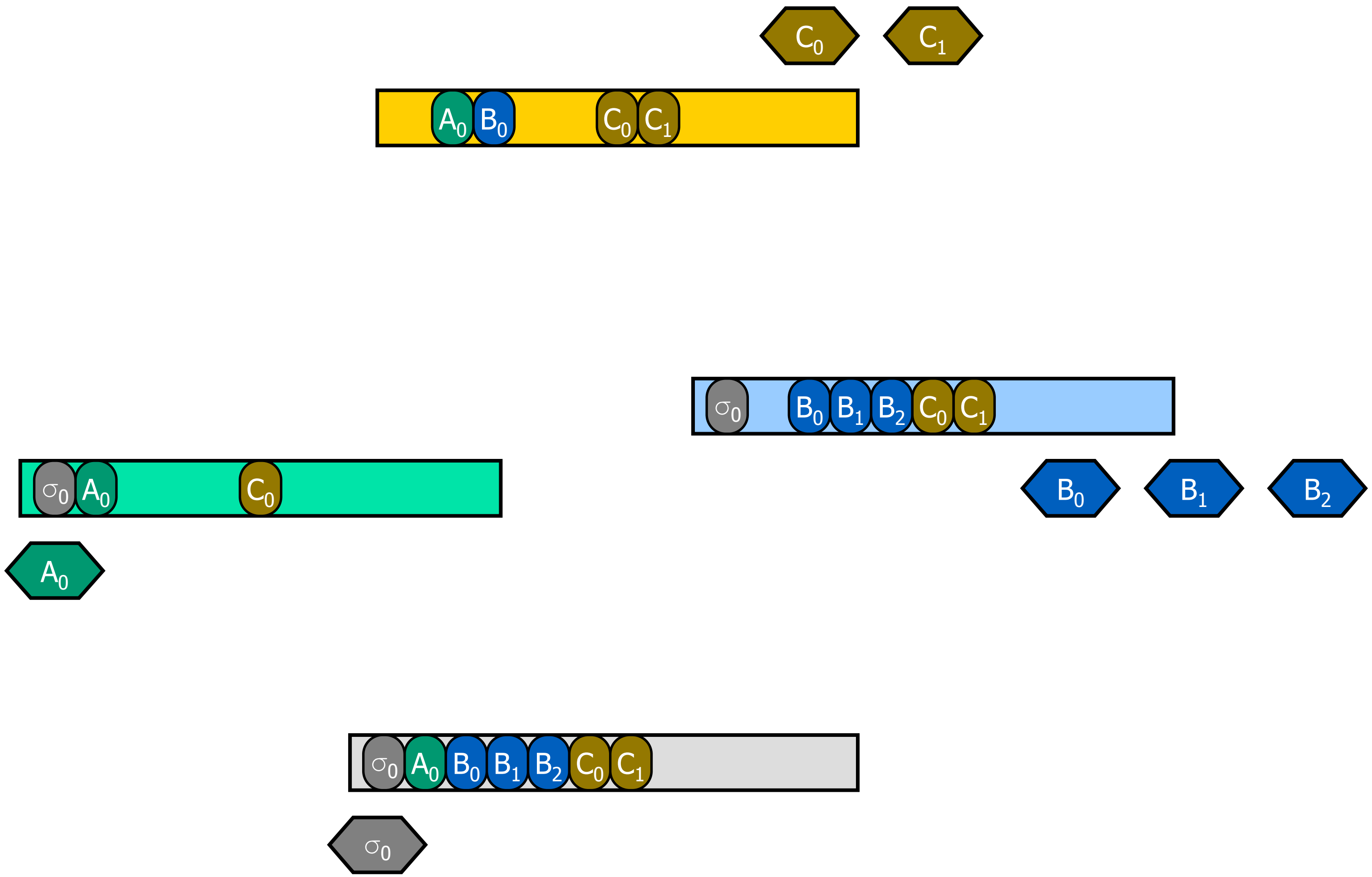


Communication Spaces

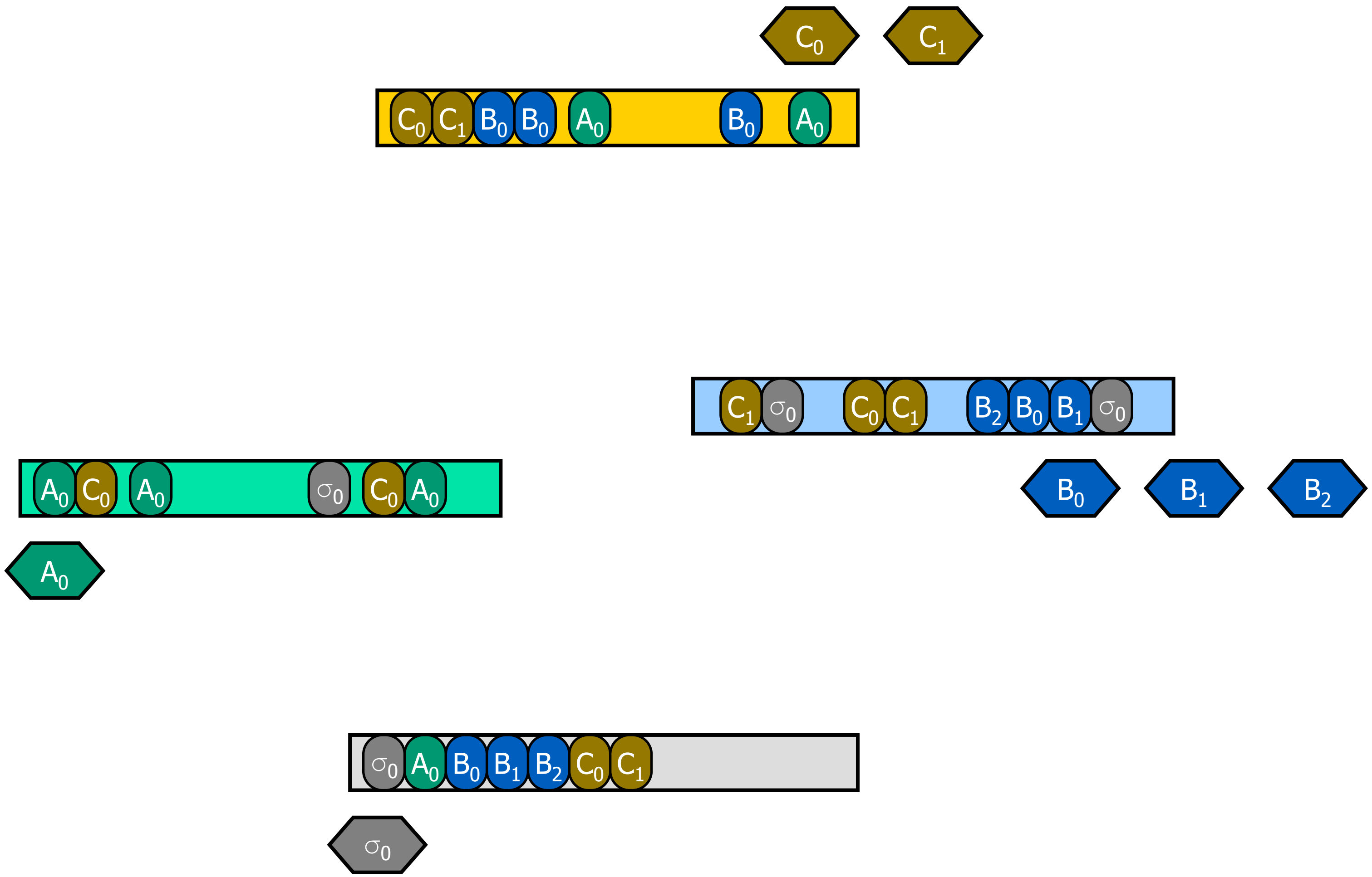
Revoking Access Rights



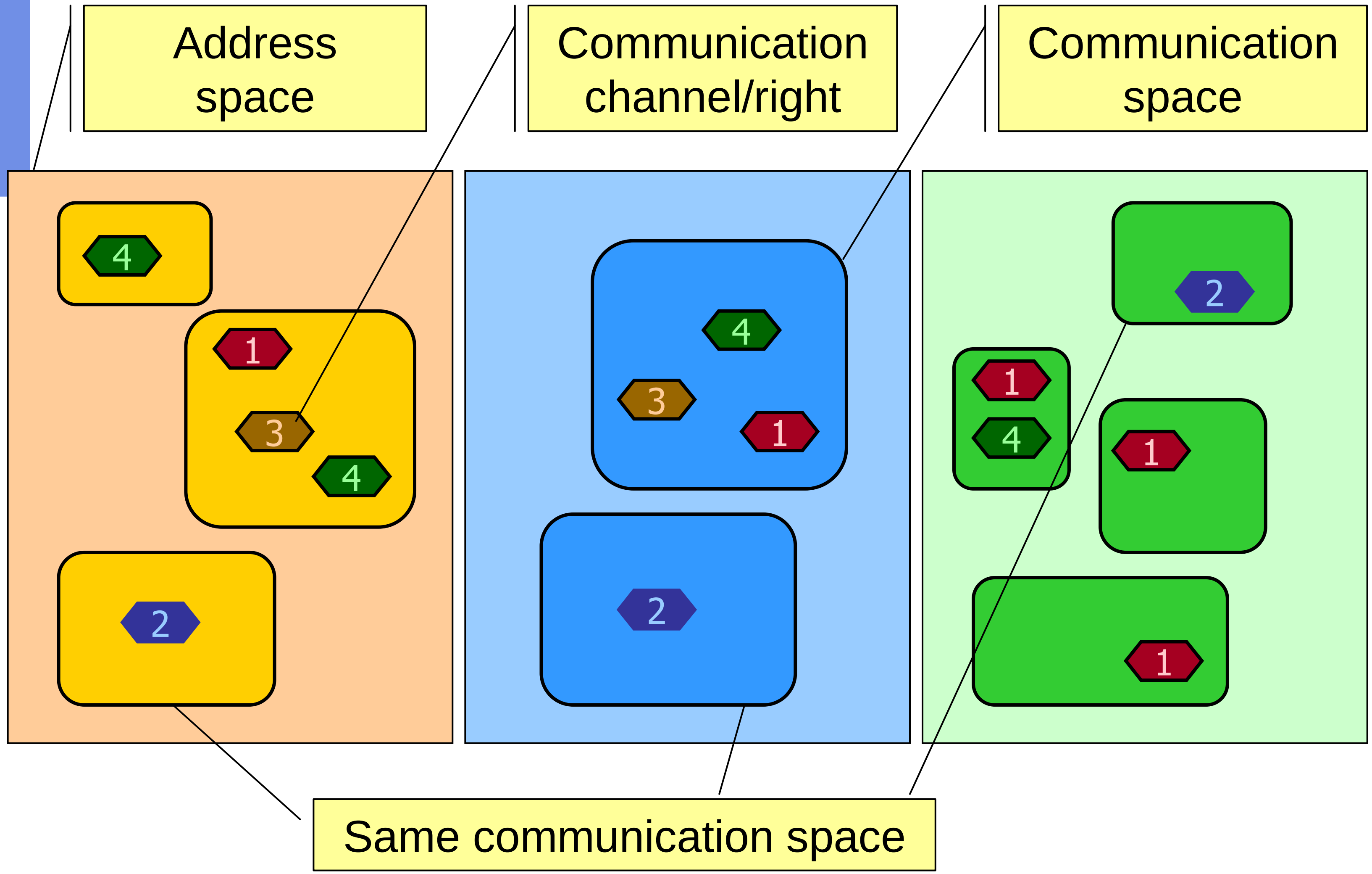
Virtual Communication Spaces
Arbitrary Thread ID Layout



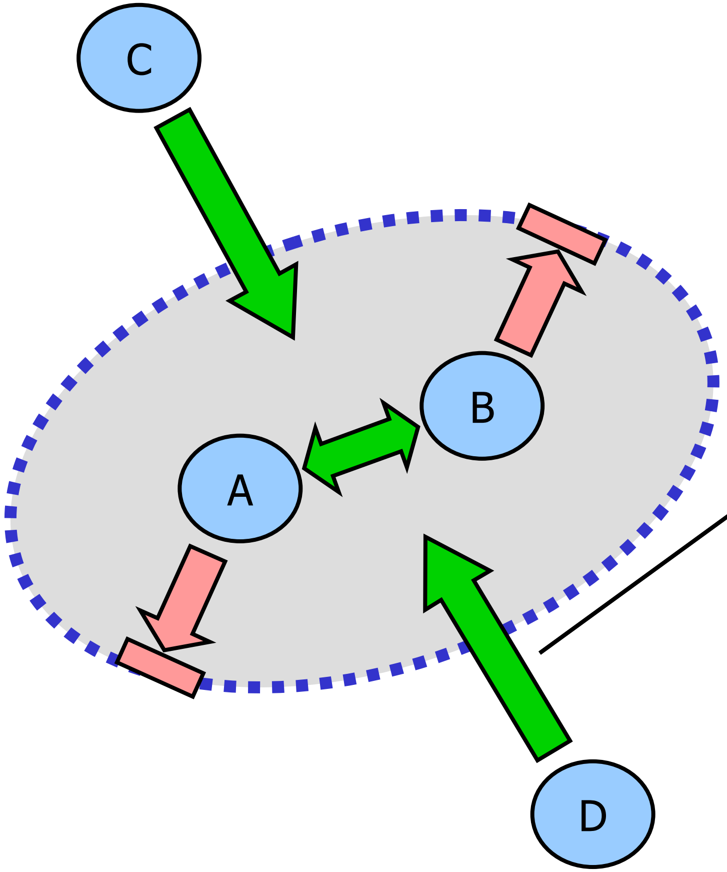
Virtual Communication Spaces
Arbitrary Thread ID Layout



Communication Spaces



Confinement (revisited)



- Can map communication rights
- Can map writable memory
- Need ability to restrict such mappings
  - Restricted via map-right

Virtual Communication Spaces
Implications on IPC Performance

- Need table lookup (indirection) to find destination thread
  - Table lookup needed anyway to check rights
- Implications of indirection for TCB lookup
  – One more cache line access per IPC
  + Smaller TLB footprint (sometimes, cf. mkc-03-aslayout)
    - TLBs usually smaller than caches
    - TLB misses often more expensive than cache misses

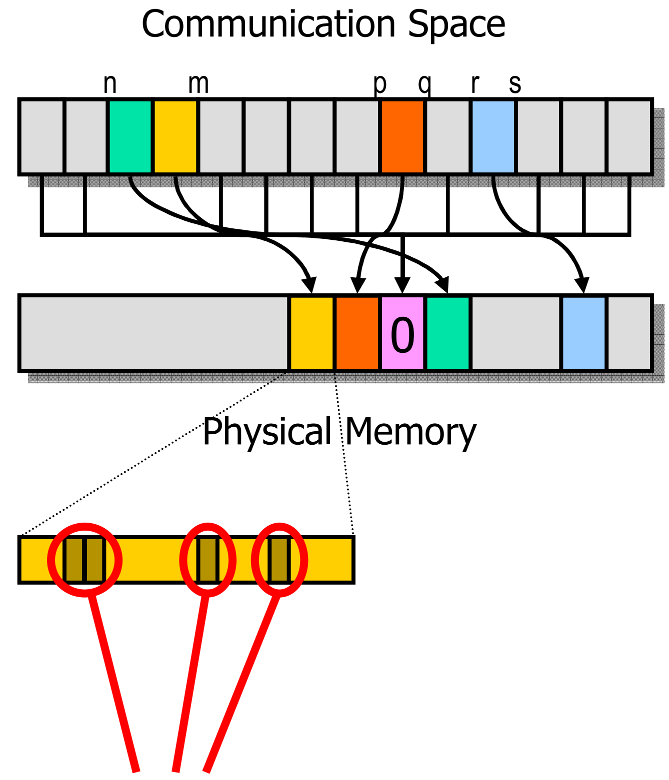
Communication Space Tables

In General

- Lookup on each IPC invocation
  - Must be extremely efficient
  - Avoid any excess indirection
  - Indirection increases
    - Cache footprint
    - Number of direct and/or indirect cache misses
- Implemented via Virtual Linear Array (VLA)
  - Lookup into dedicated virtual memory area
  - Area with a valid mapping backed by dedicated page frame
  - Area with no valid mapping backed by zero page
    - All read accesses return zero
    - Cf. 0-mapping trick

Communication Space Tables

Virtual Linear Array

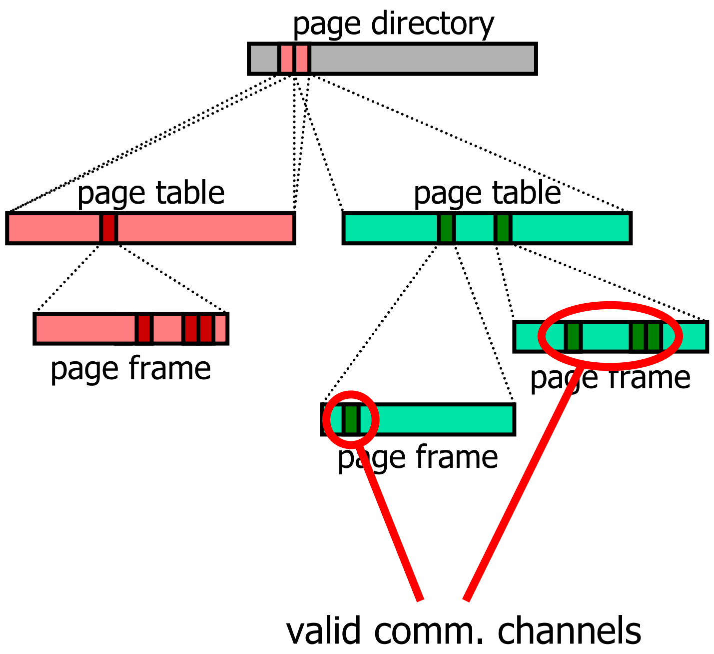


- Implemented via Virtual Linear Array (VLA)
  - Lookup into dedicated virtual memory area
  - Area with a valid mapping backed by dedicated page frame
  - Area with no valid mapping backed by zero page
    - All read accesses return zero
    - Cf. 0-mapping trick

Communication Space Tables

Multiplexing Comm. Space Areas

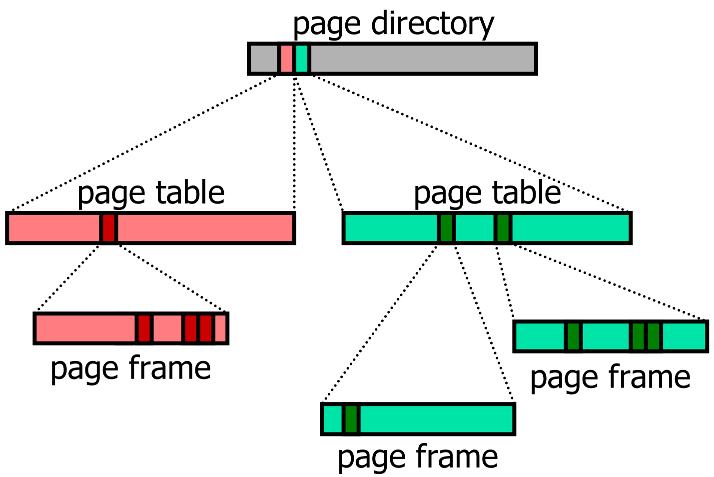
- Memory space may span multiple comm. spaces
  - Per thread comm. space
- Solutions
  - Rewrite page table on thread switch
    - Also requires TLB flush
  - Keep multiple comm. spaces in page table
    - Keep pointer to appropriate area



Communication Space Tables

Comm. Space Areas and Untagged TLBs

- IA-32: Switching page tables also flushes TLB
  - All comm. space TLB entries flushed
  - TLB misses costly
- Global TLB entries
  - Not flushed on page table reload
  - Dedicate comm. spaces to fixed VM areas
  - Mark all TLB entries global



Communication Space Tables

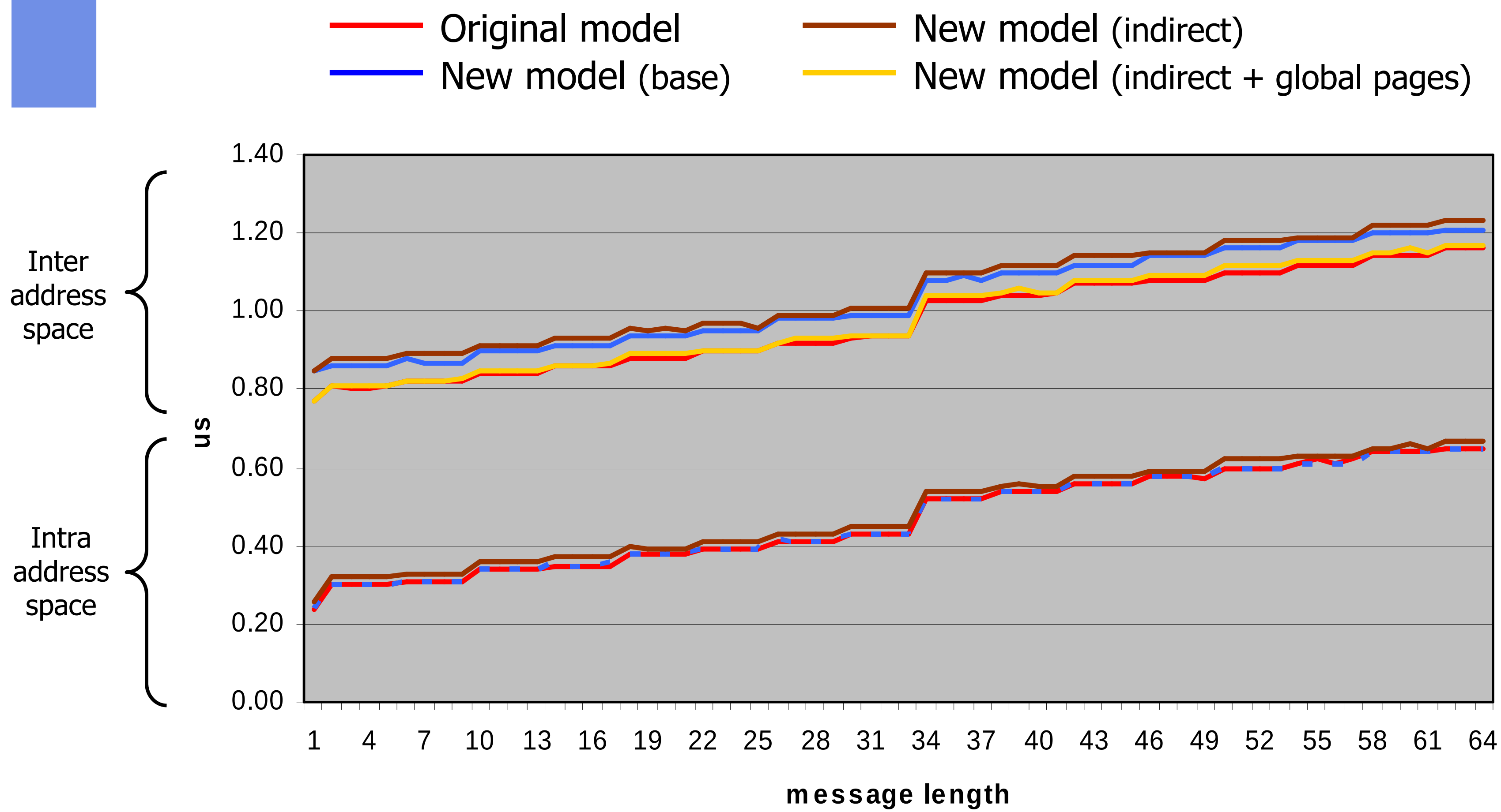
Virtual Memory Requirements (IA-32)

- Comm. channel descriptor requires
  - TCB pointer
  - Access rights
  - Receive descriptor
  - Pointer to map node
- One descriptor requires 8 bytes
  - 4 MB can hold 512k channels

• Encoded in 2 words

• Stored in shadow page
• Shadow page not mapped into virtual memory

IPC Performance
Pentium III, 500 MHz, Non-MP Kernel



IPC Resource Consumption

Pentium III, 32 Byte Cache Lines, Non-MP Kernel

Original Model

	Memory		Cache		TLB
	Rd	Wr	Rd	Wr	
TCB	2/11	3/1	2/1	1/1	1/1
UTCB	0/3	0/1	0/1	0/1	0/0
Com	-	-	-	-	-
Total	5/14	3/2	2/2	1/2	1/1
	8/16		3/4		

New Model

	Memory		Cache		TLB
	Rd	Wr	Rd	Wr	
TCB	4/8	5/1	1/1	1/1	0/0
UTCB	1/3	0/1	1/1	0/1	0/0
Com	2/0	0/0	1/0	0/0	1/0
Total	7/11	5/2	3/2	1/2	1/0
	12/13		4/4		

Stack references not taken into account (same for both kernels)

(send phase/receive phase)

All Access is via IPC (revisited)

- What microkernel mechanisms are needed for security?
  - How do we authenticate?
    - Sender's ID revealed on IPC
    - Sender ID is unforgeable

All Access is via IPC (revisited)

- What microkernel mechanisms are needed for security?
  - How do we authenticate?
  - How do we perform authorization?
    - Give thread rights to communicate via mappings
    - Revoke rights to communicate via unmap
    - Individual servers can decide on fine grained policies

All Access is via IPC (revisited)

- What microkernel mechanisms are needed for security?
  - How do we authenticate?
  - How do we perform authorization?
  - How do we implement arbitrary security policies?
    - Authorization performed completely in user-level

All Access is via IPC (revisited)

- What microkernel mechanisms are needed for security?
  - How do we authenticate?
  - How do we perform authorization?
  - How do we implement arbitrary security policies?
  - How do we enforce arbitrary security policies?
    - Any communication requires the appropriate communication right