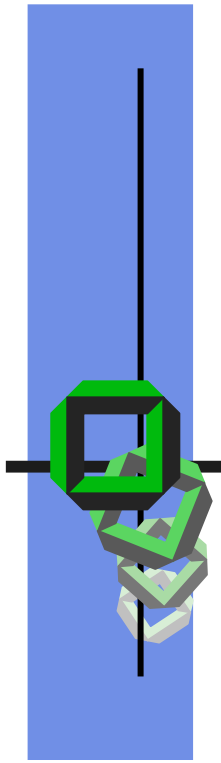
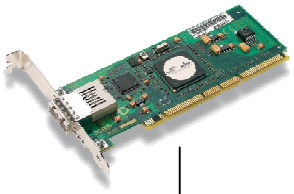
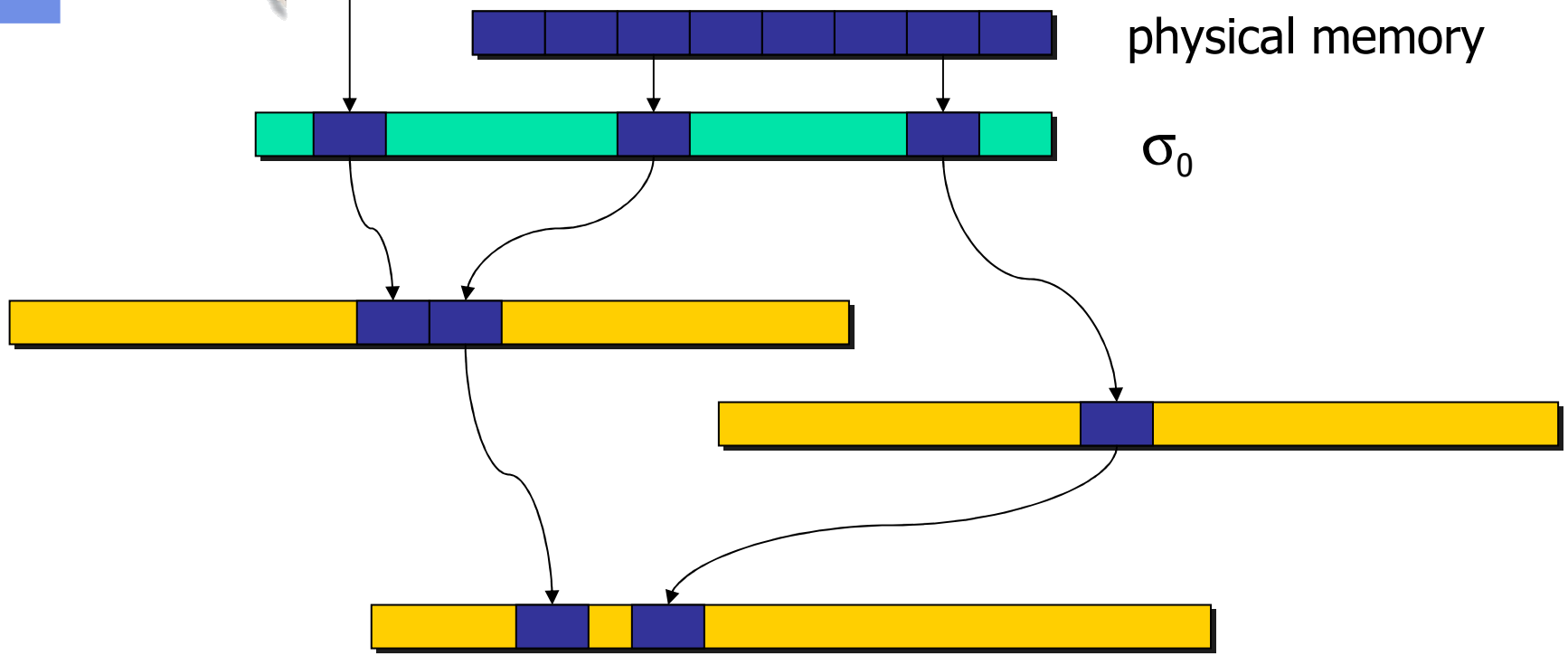


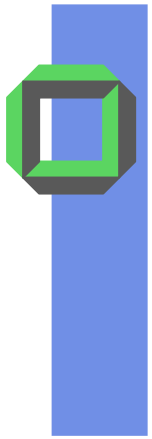
µ-Kernel Construction (7)

Virtual Memory Mapping

Address Space Construction





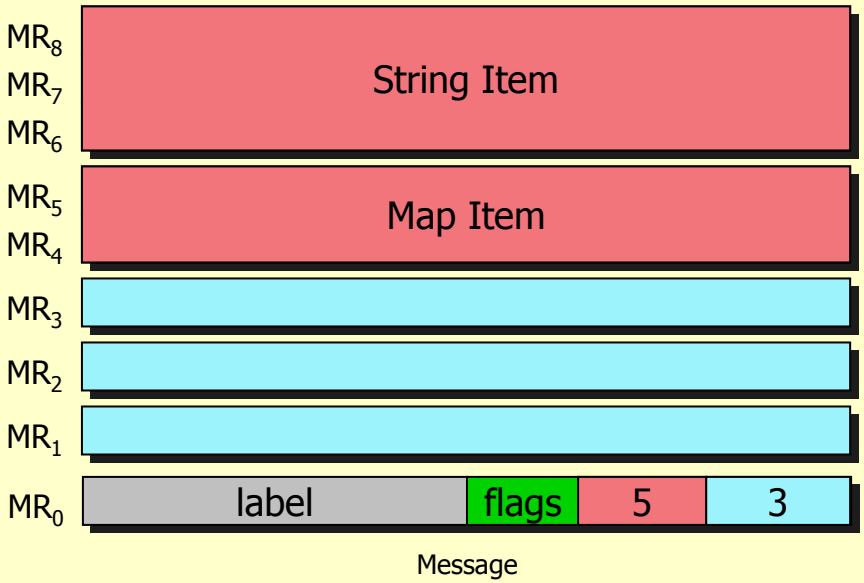


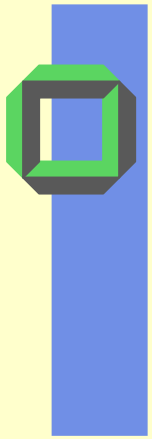
Mechanisms

- Address space construction
  - Map [populate]
  - Unmap [wipe out]
- Integrity and security
  - Access permissions [rwx]
- Resource control
  - Use bits [accessed or dirty]
  - Page fault messages [detect page use]

Review: Message Construction

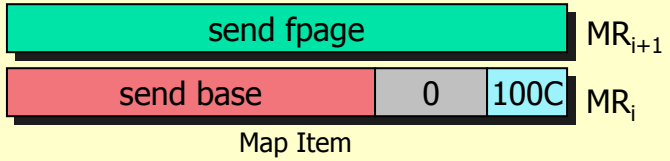
- Typed items occupy one or more words
- Three currently defined items
  - Map item (2 words)
  - Grant item (2 words)
  - String item (2+ words)
- Typed items can have arbitrary order

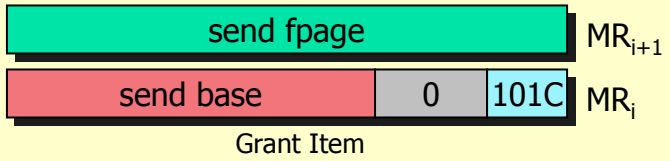


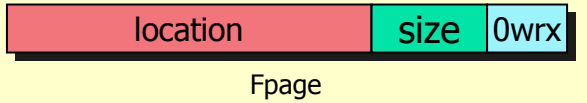


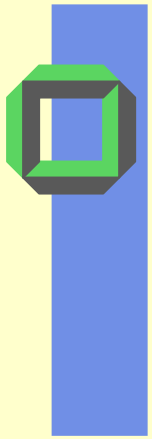
Review: Map and Grant Items

- Two words
  - Send base
  - Fpage
- Lower bits of send base indicates map or grant item





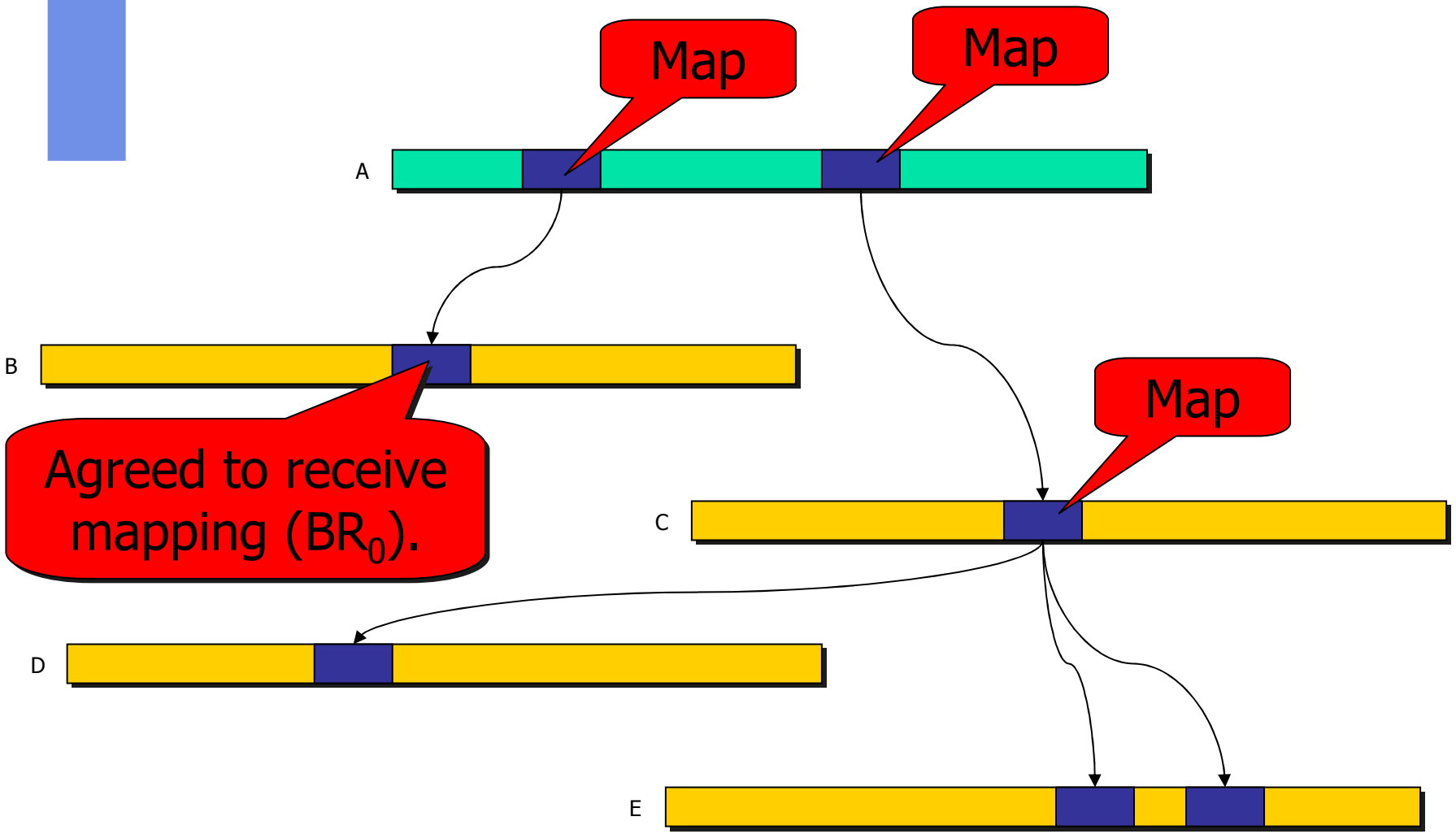




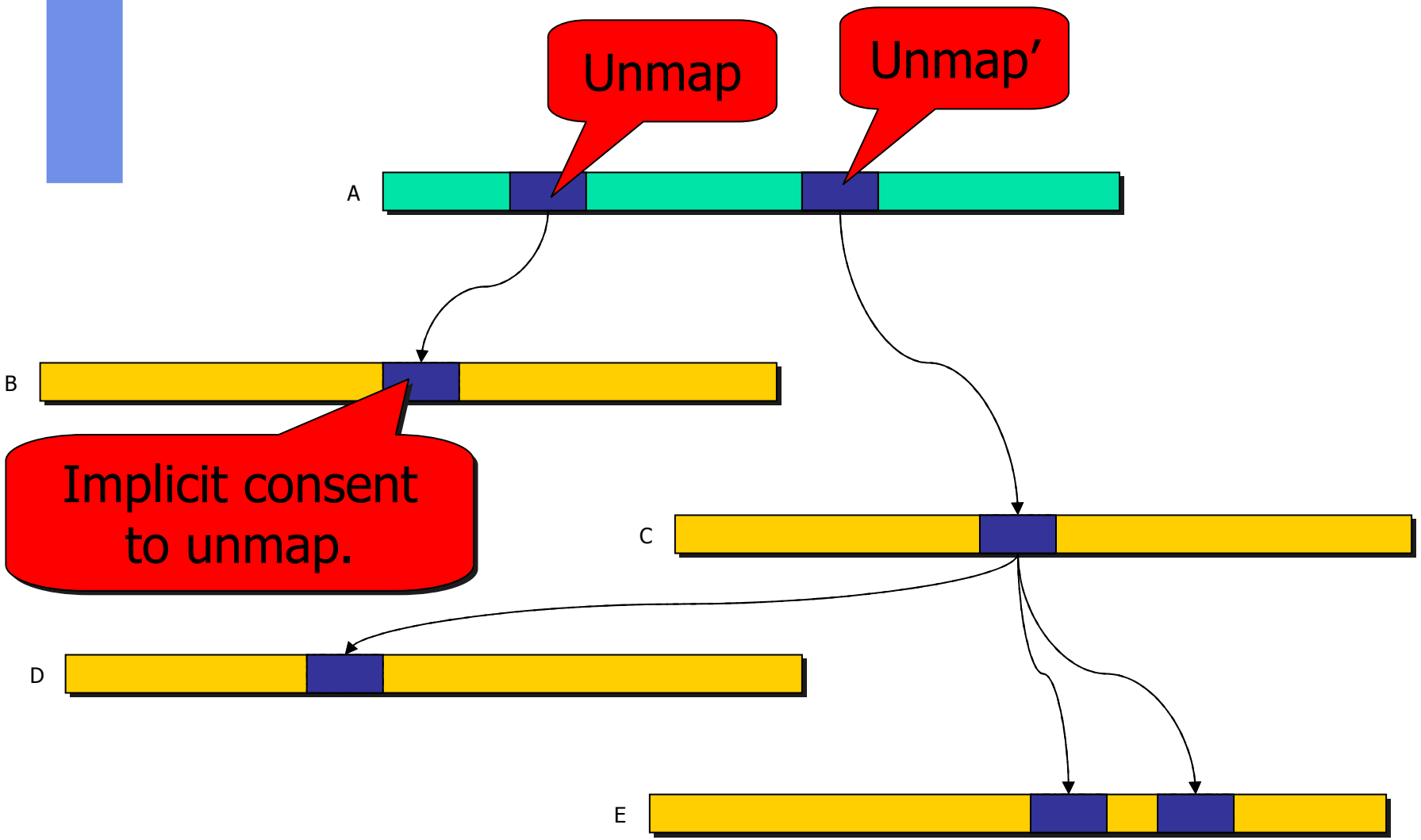
Review: Receiving Messages

- Receiver buffers are specified in registers (BR_0 ... BR_{33})
- First BR (BR_0) contains "Acceptor"
  - May specify receive window (if not nil-fpage)
  - May indicate presence of receive strings/buffers (if s-bit set)

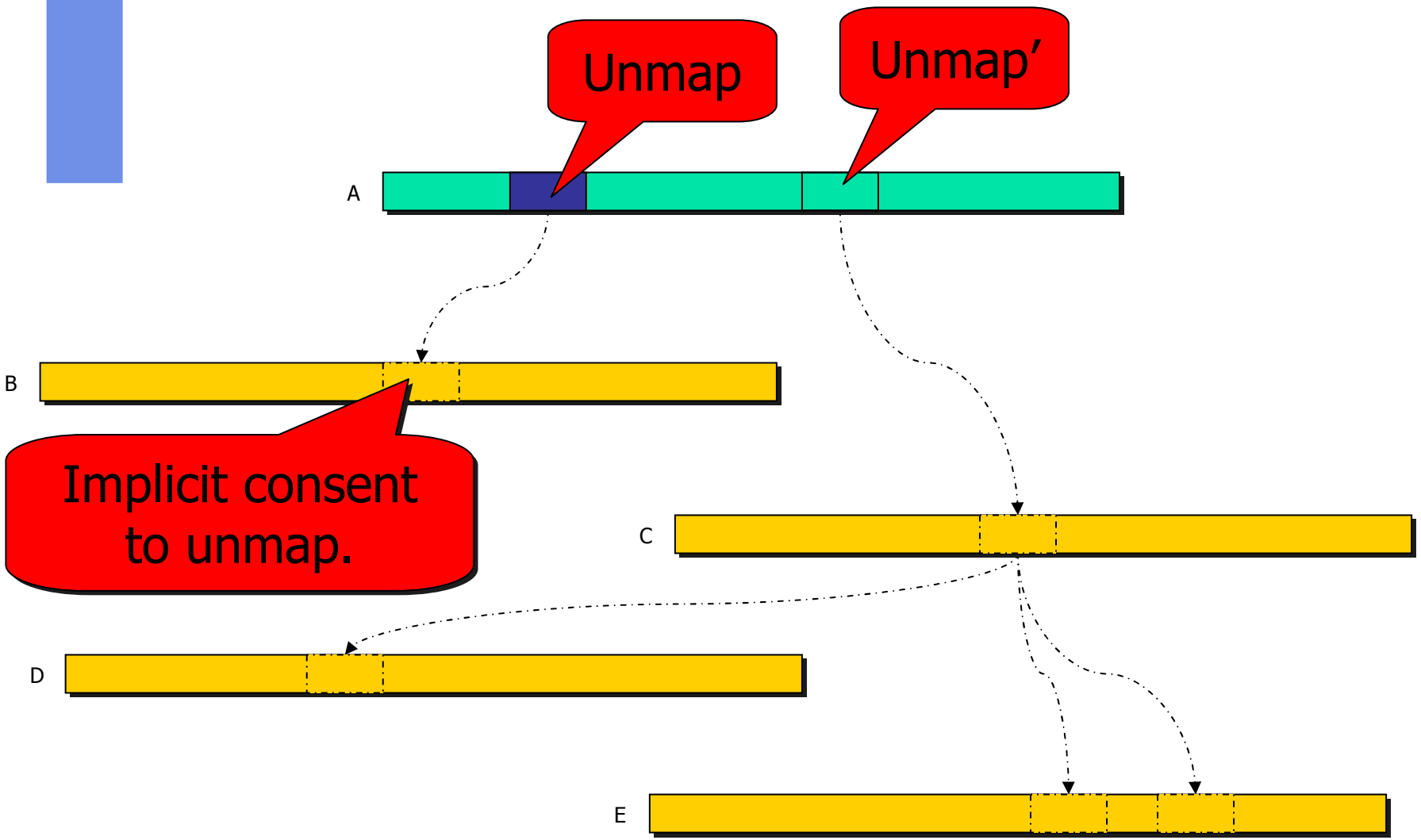
Map



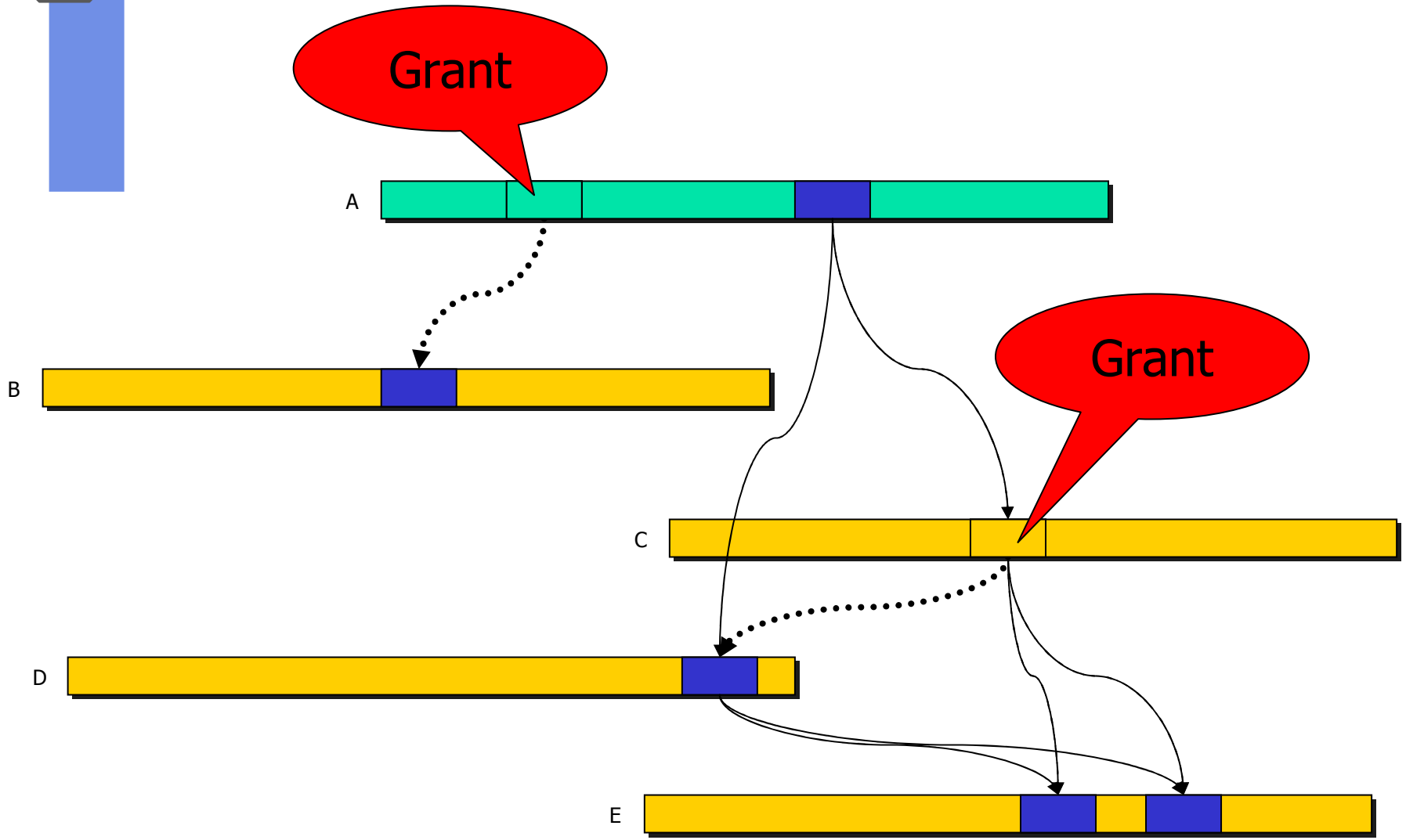
Unmap



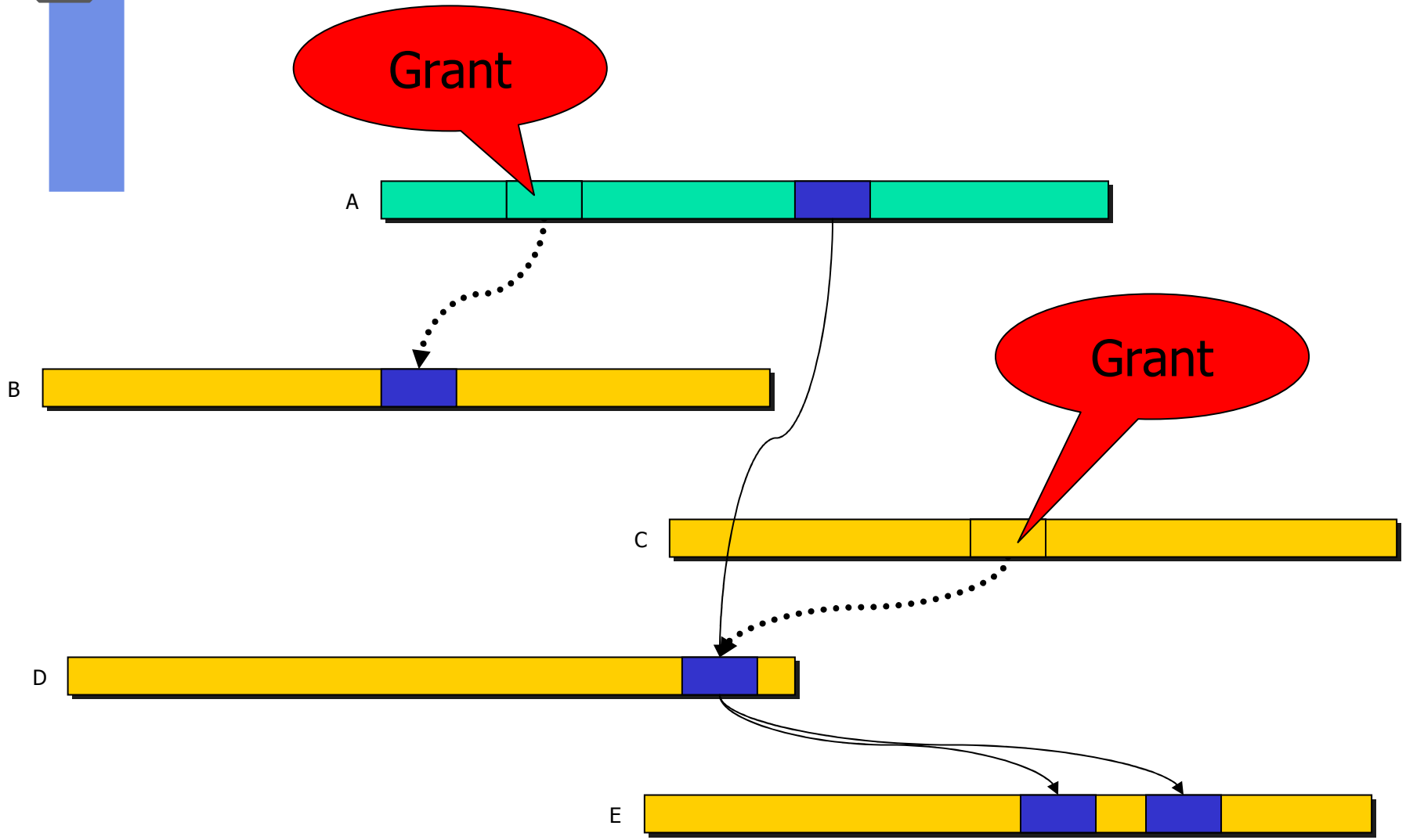
Unmap



Grant

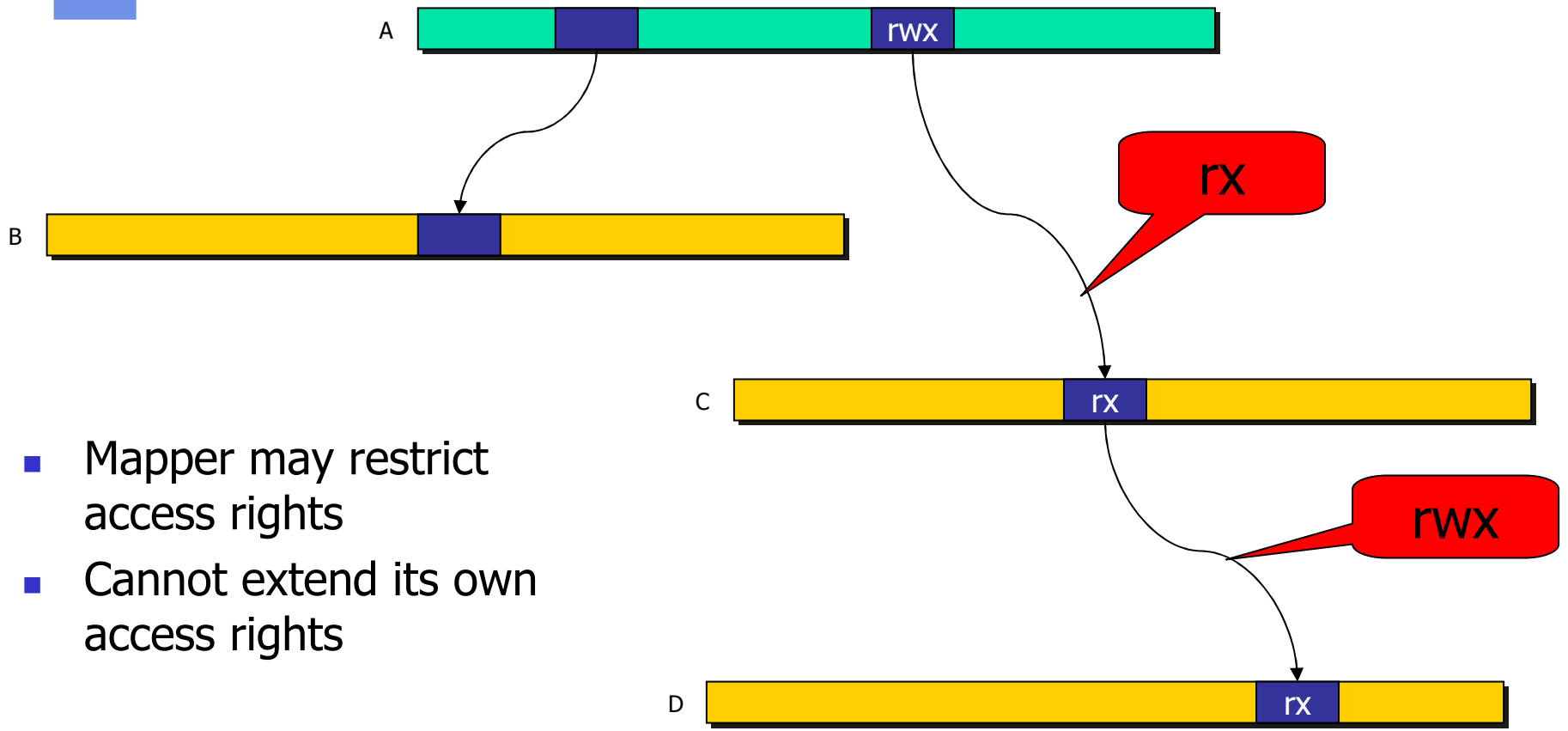


Grant



Access Rights – Map

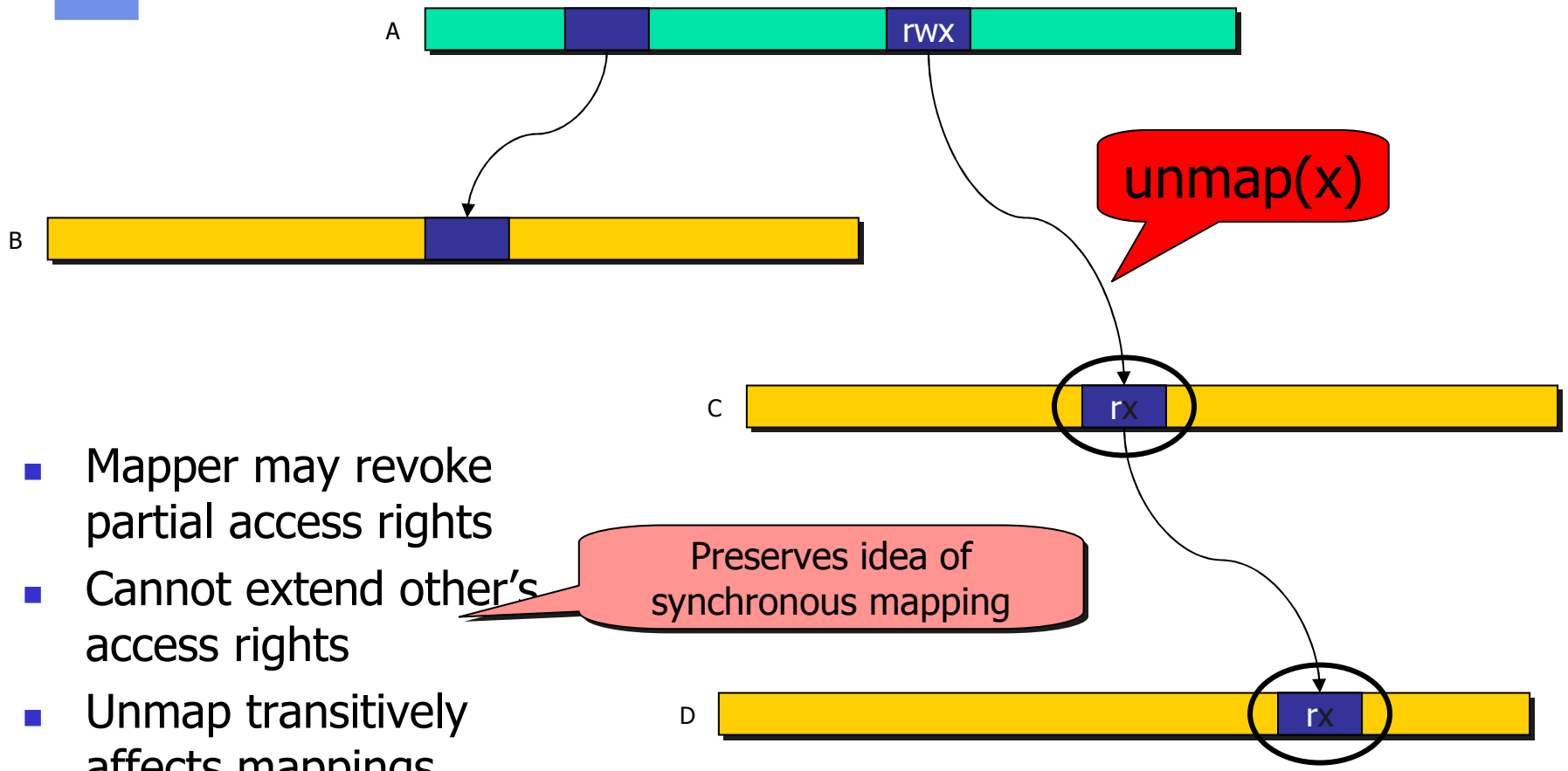
r = Read
w = Write
x = eXecute



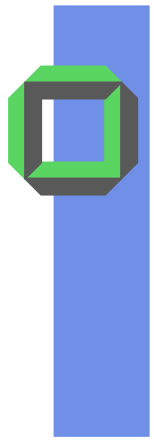
- Mapper may restrict access rights
- Cannot extend its own access rights

Access Rights – Unmap

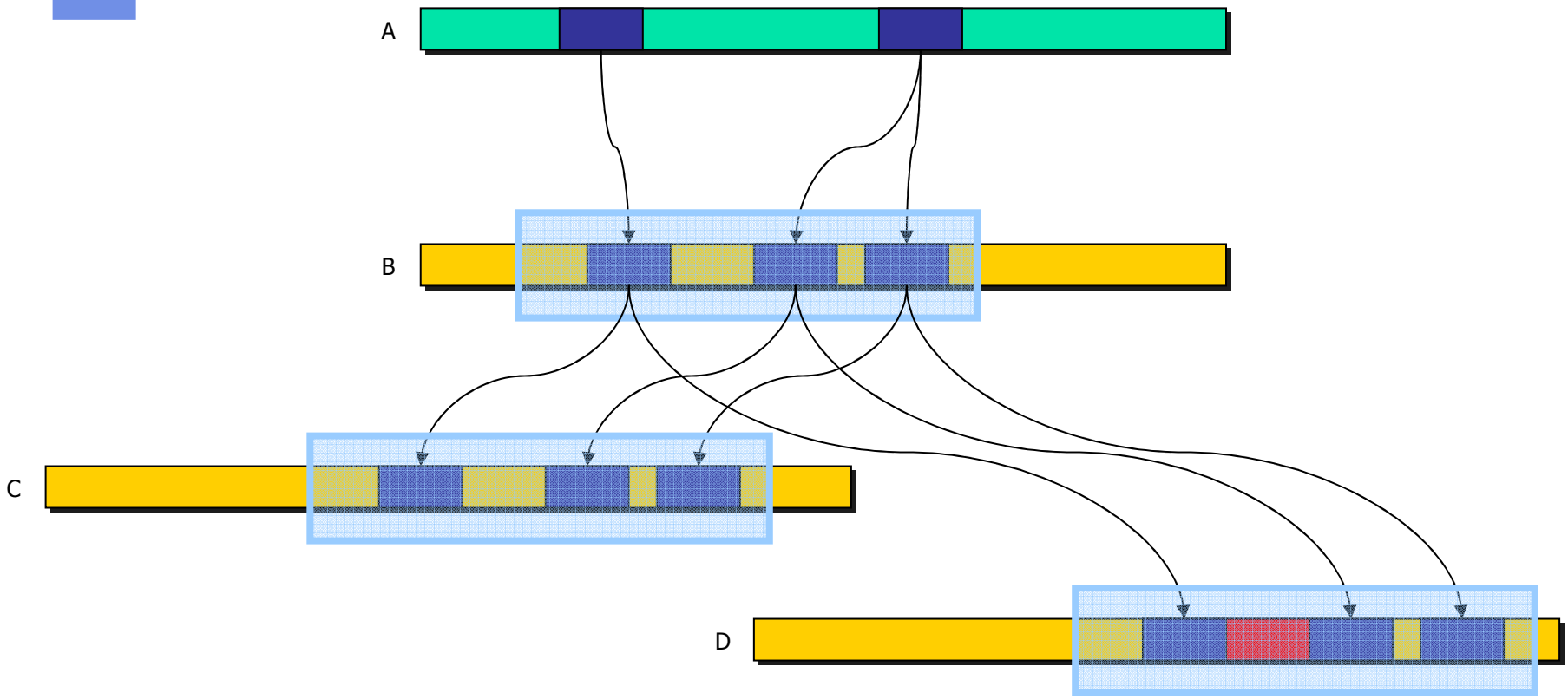
r = Read
w = Write
x = eXecute

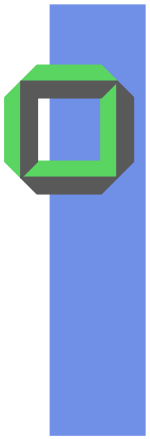


- Mapper may revoke partial access rights
- Cannot extend other's access rights
- Unmap transitively affects mappings



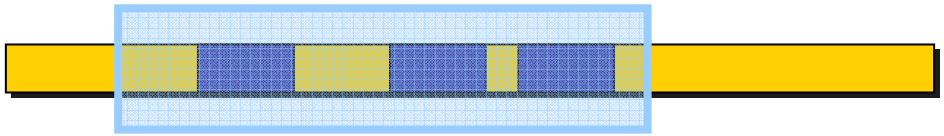
Mapping Regions



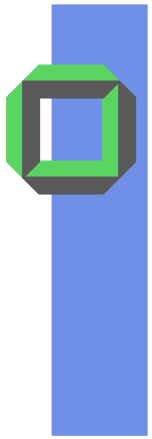


Mapping Regions: Flex Pages

- Abstraction: flex page
  - Contiguous regions of virtual address space
    - Sparse physical mappings possible
  - Called **fpage**
  - Abstracts architecture's page sizes



- Fpage semantics
  - Inseparable object
  - Aligned to its size
  - Size is power of 2, min. $1024=2^{10}$ byte



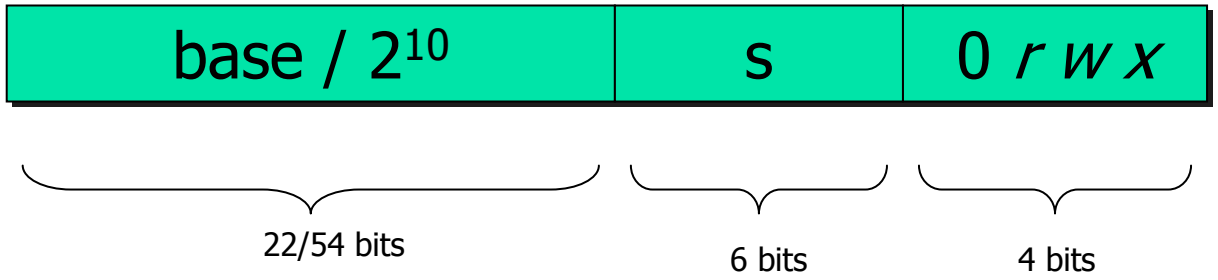
Fpage Encoding

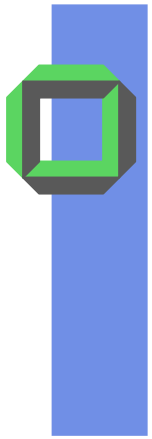
- Special cases
  - Complete address space (base=0, s=1)
  - Nothing: nilpage (0)

fpage(base, size=2^s)

$s \geq 10$

base mod $2^s = 0$



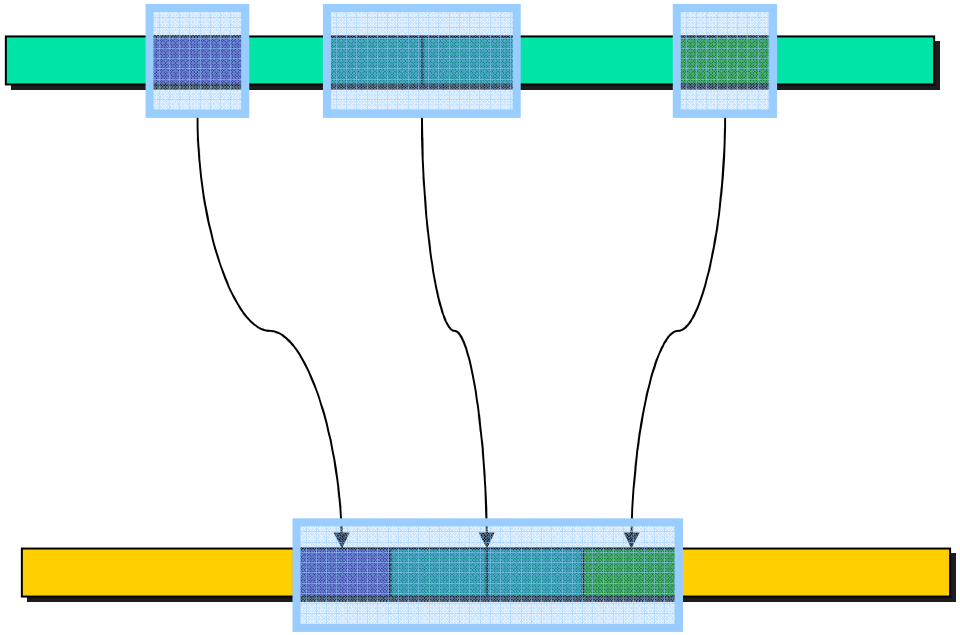


Mapping Fpages

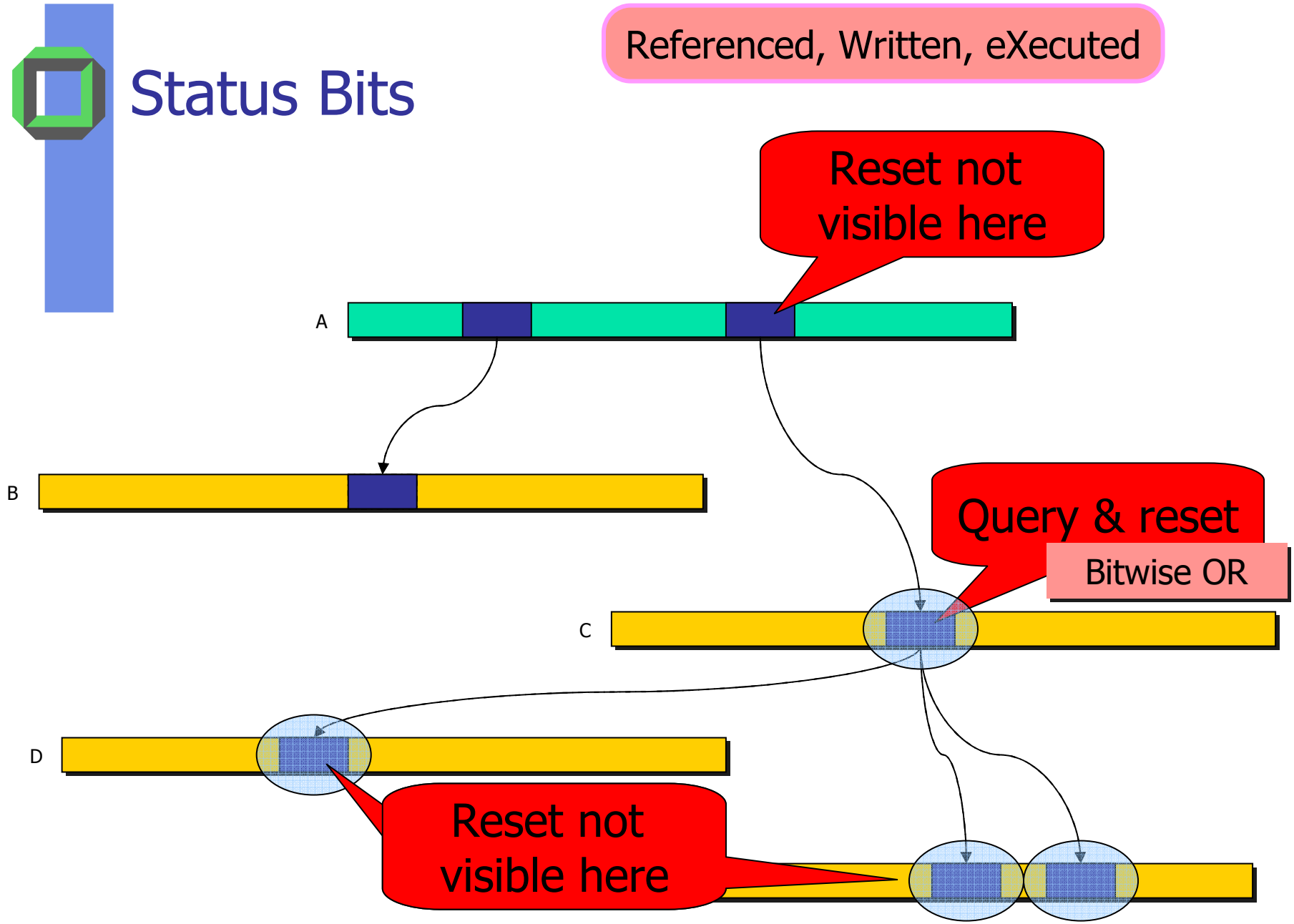
- Receive window too large?
  - Use offset (aka send base)
- Common case: page faults
  - Receive fpage is entire address space
  - Send offset is fault address

Map($fpage_1$ to $offset_1$
$fpage_2$ to $offset_2$
$fpage_3$ to $offset_3$)

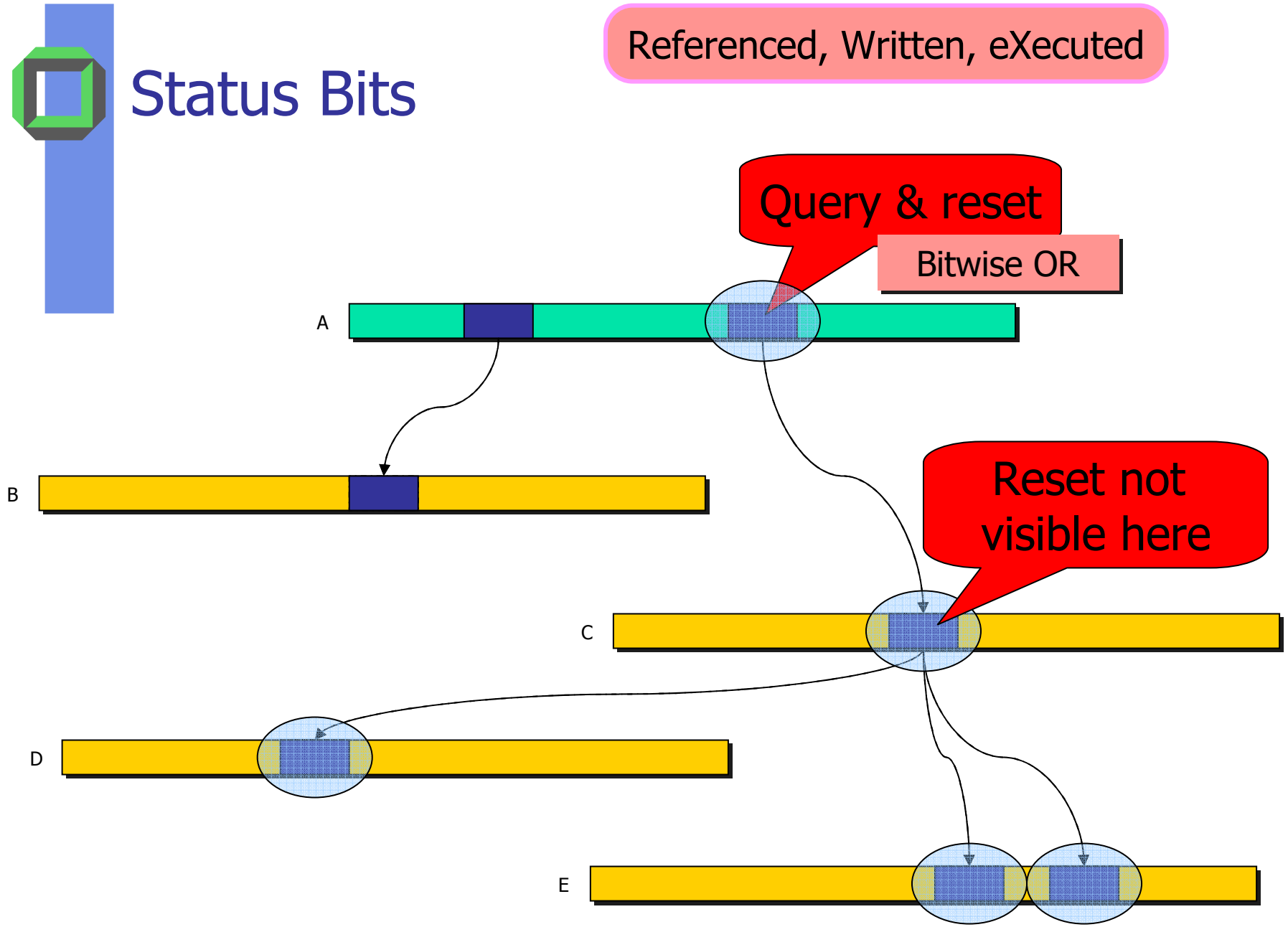
MapAccept(fpage)

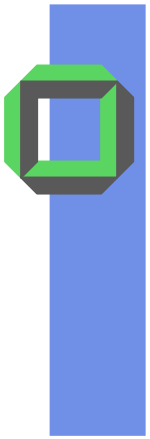


Status Bits



Status Bits

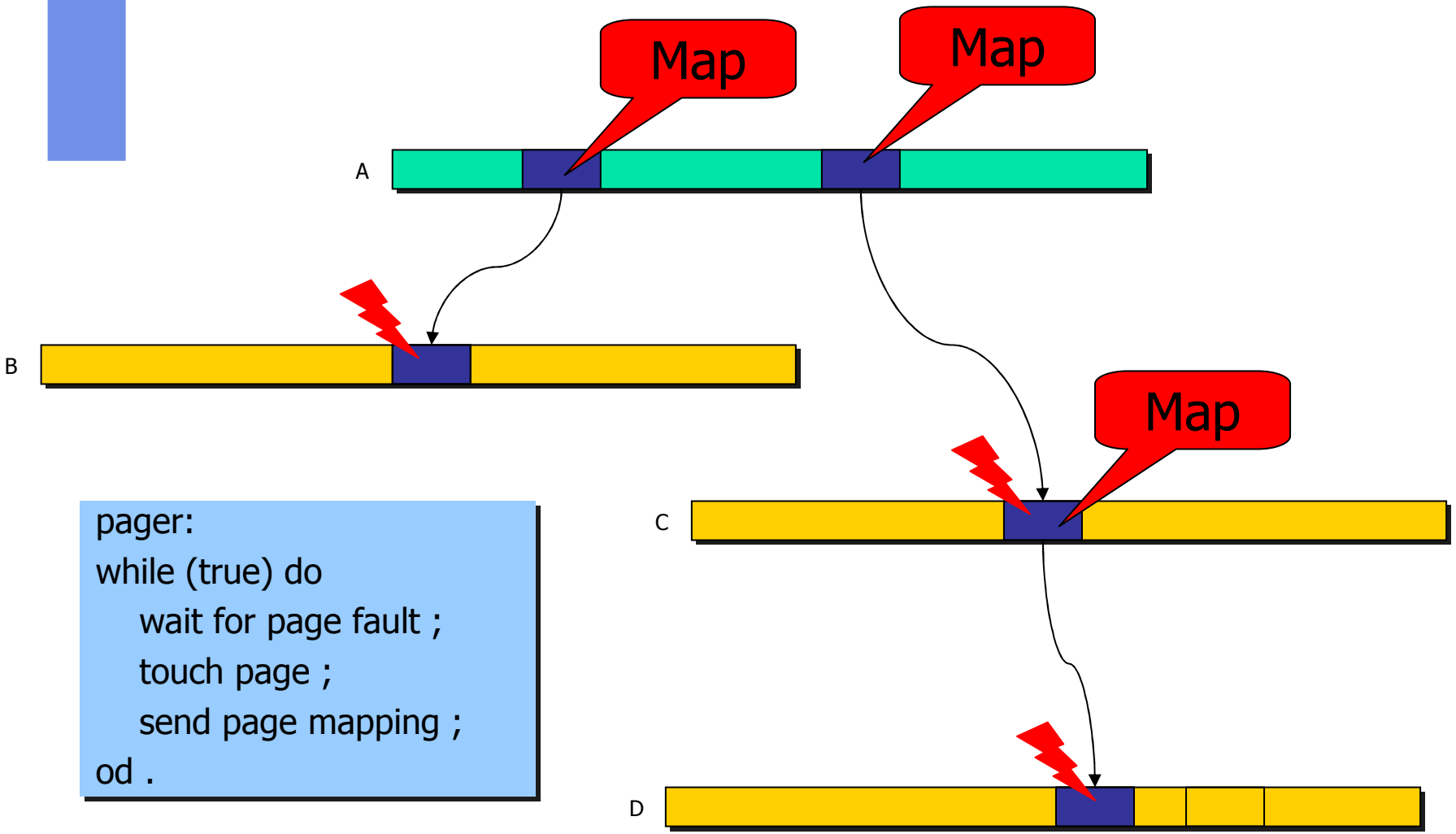




Resource Control – Not Enough Memory

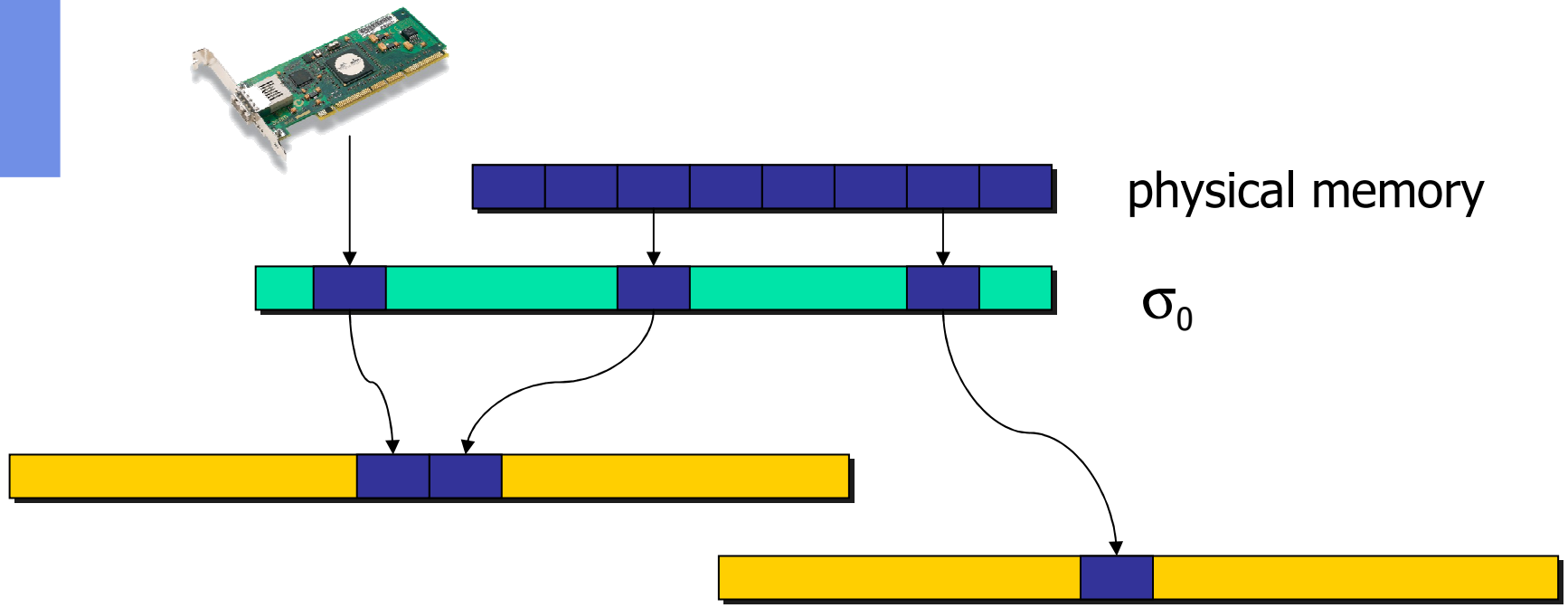
1. Find set of pages X to swap to disk
   - Check status bits
2. Unmap X from idle client C
3. Write X to disk
4. Map X to hungry client
5. What happens when C accesses X?
   1. Page fault
   2. Pager repeats algorithm, but to satisfy C

Page Miss Handling

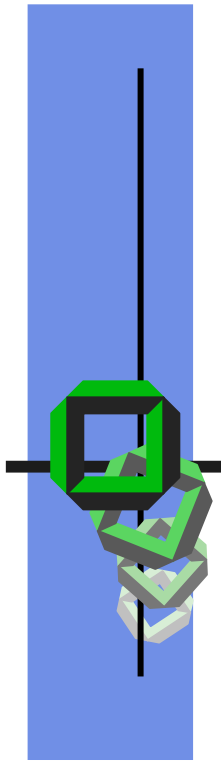



```
pager:
while (true) do
    wait for page fault ;
    touch page ;
    send page mapping ;
od .
```

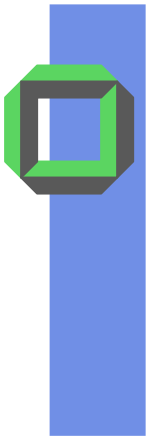
Initial Address Space



- Idempotent mappings to sigma0
  - Virtual address = physical address
- User-level knows *all* virtual to physical mappings
  - Necessary for DMA



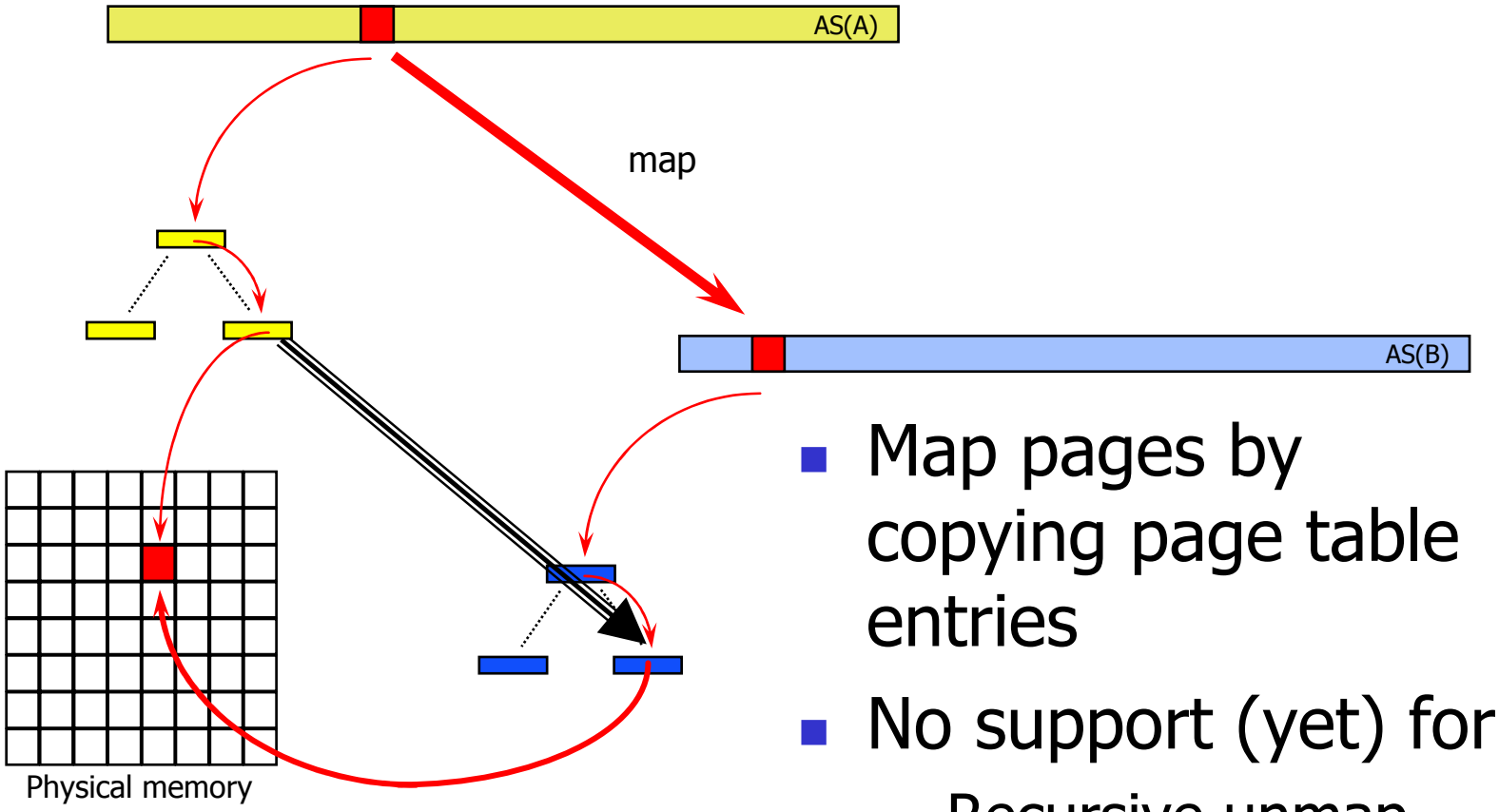
Implementation



Mapping Regions

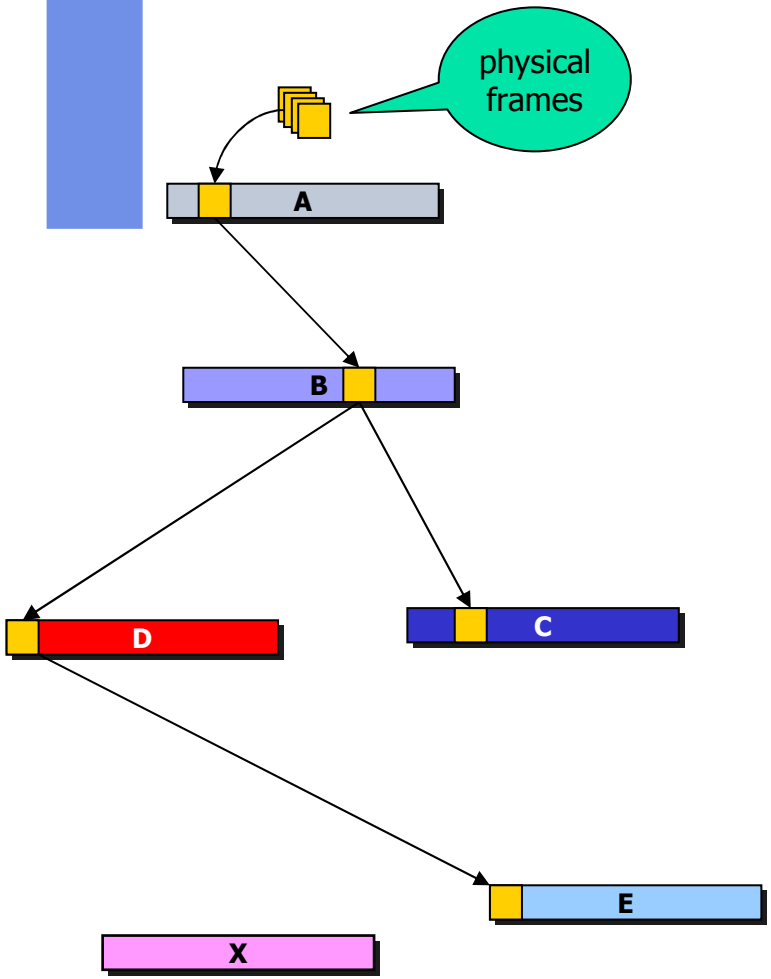
- Implementation
  - Based on page tables
  - Physical page (frame)
    - Basic mapping unit
    - Determines minimum alignment
- Minimum fpage size
  - Physical page size

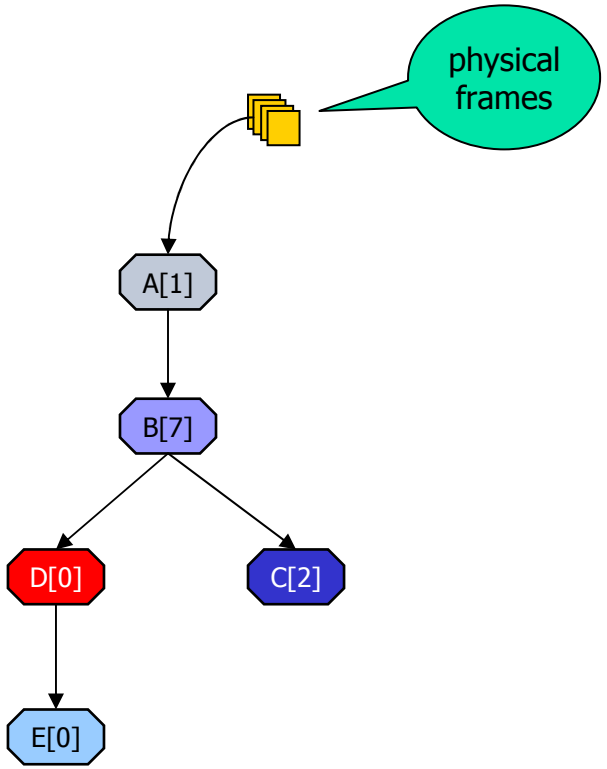
Mapping Pages



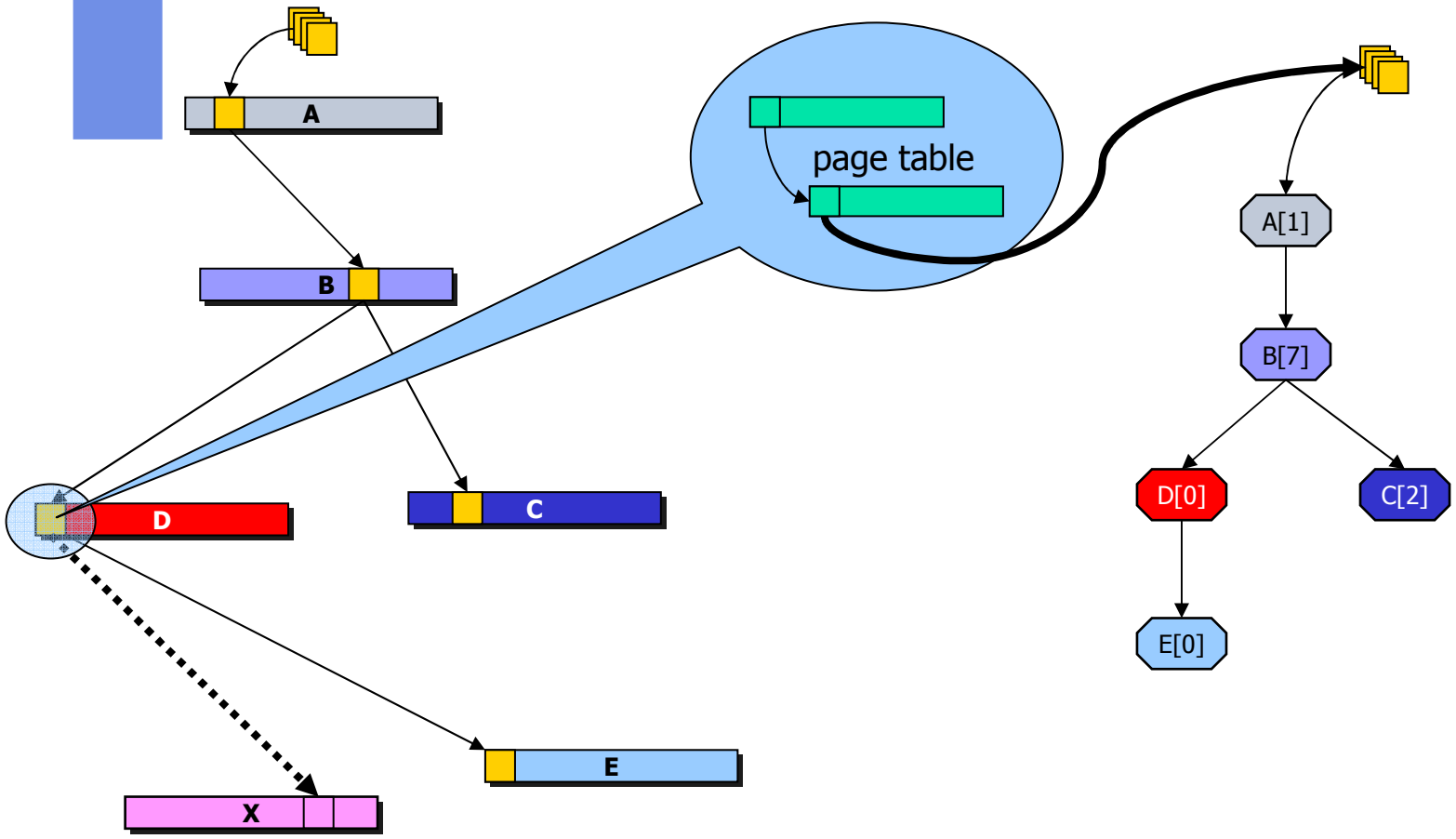
- Map pages by copying page table entries
- No support (yet) for
  - Recursive unmap
  - Combined status bits

Mapping Database

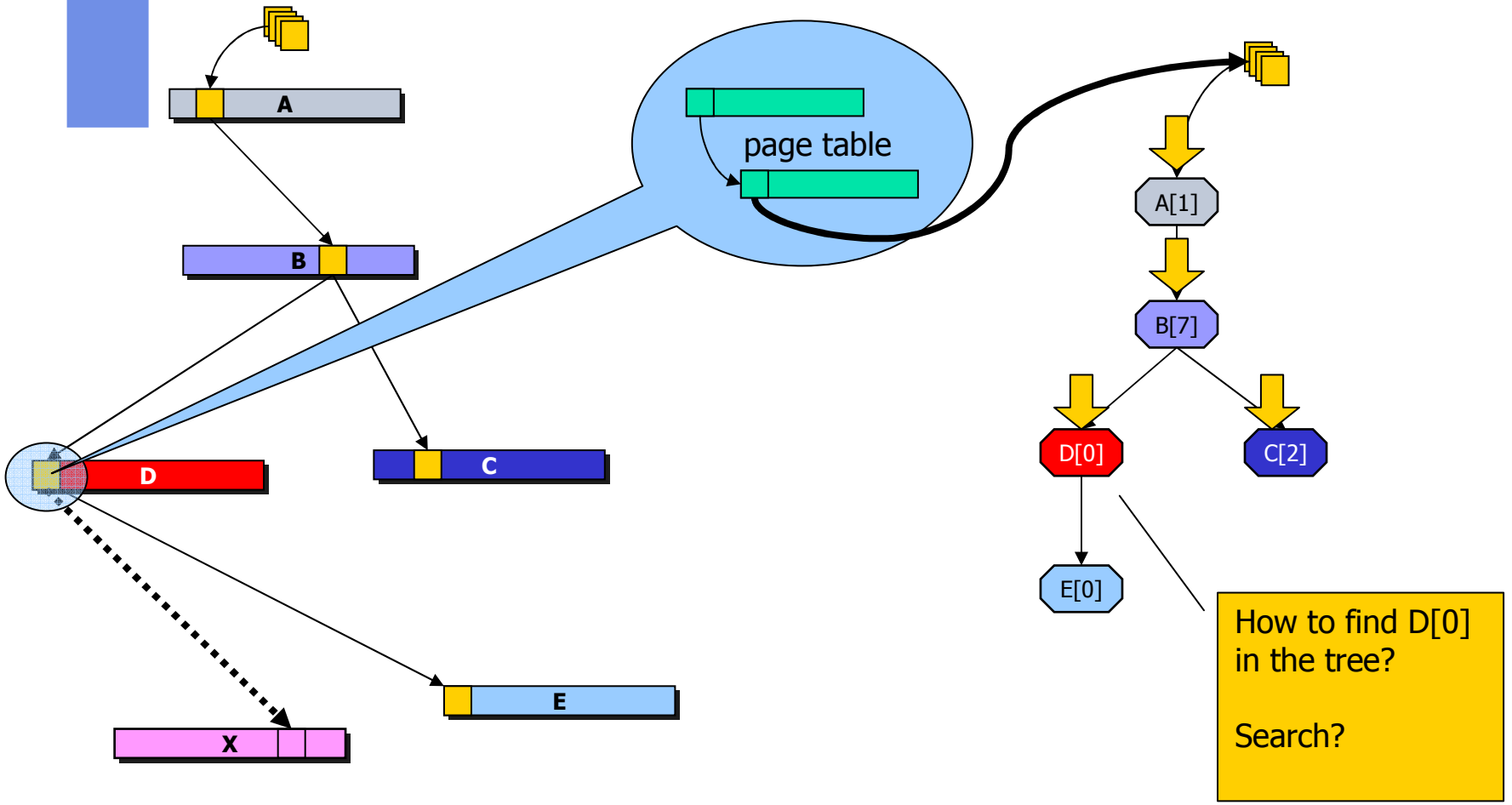




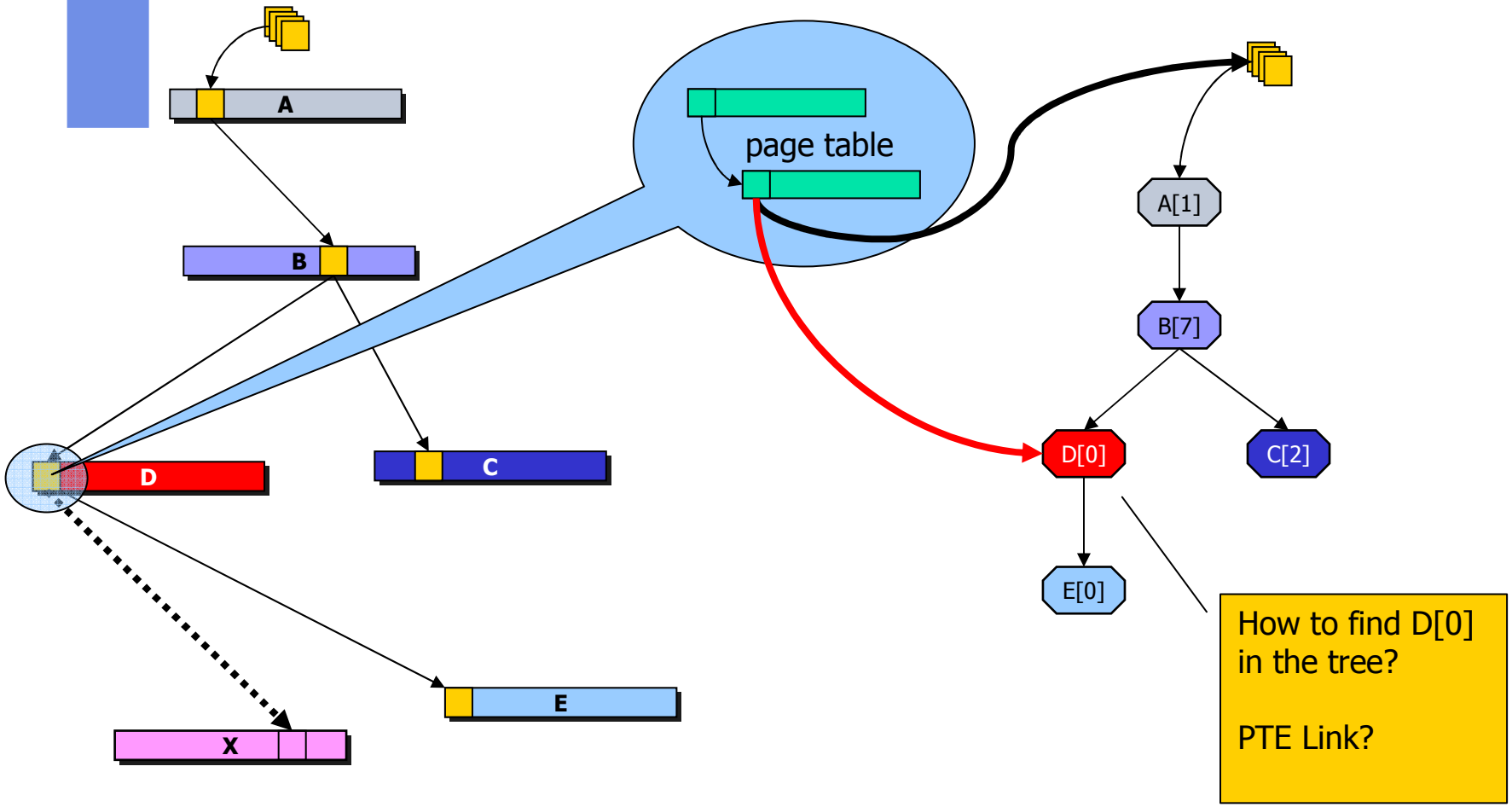
Mapping Database – Create Mapping



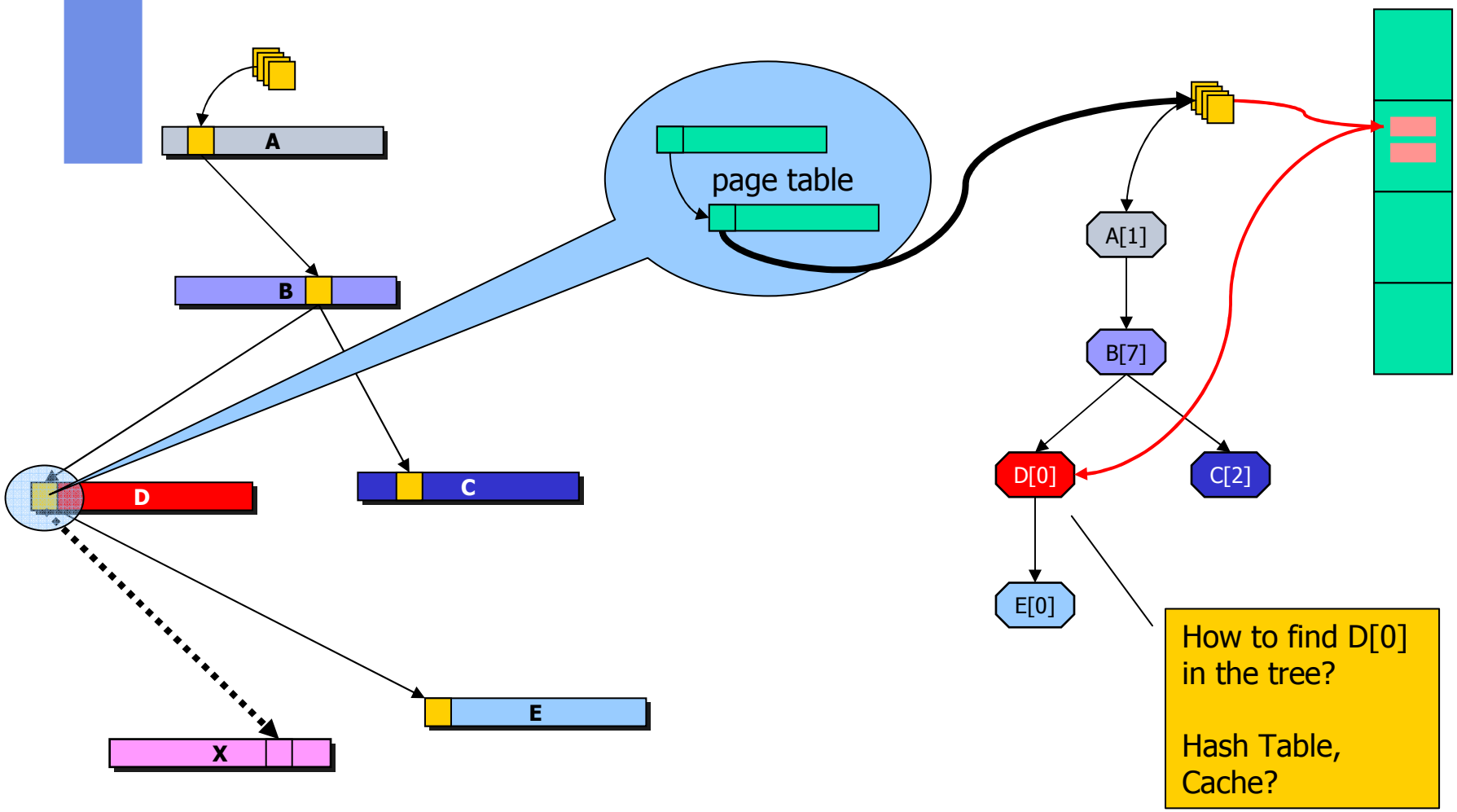
Mapping Database – Create Mapping



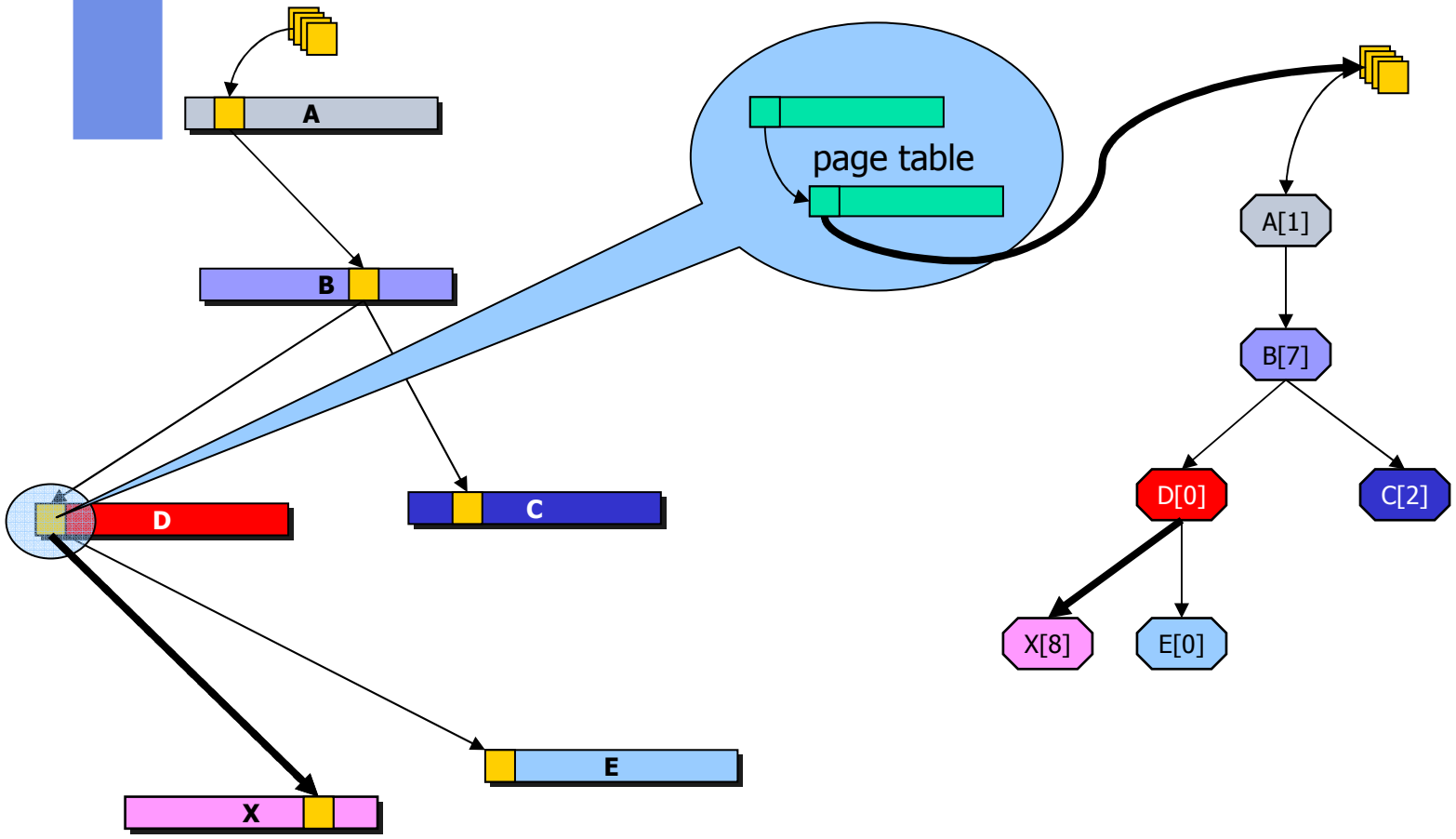
Mapping Database – Create Mapping



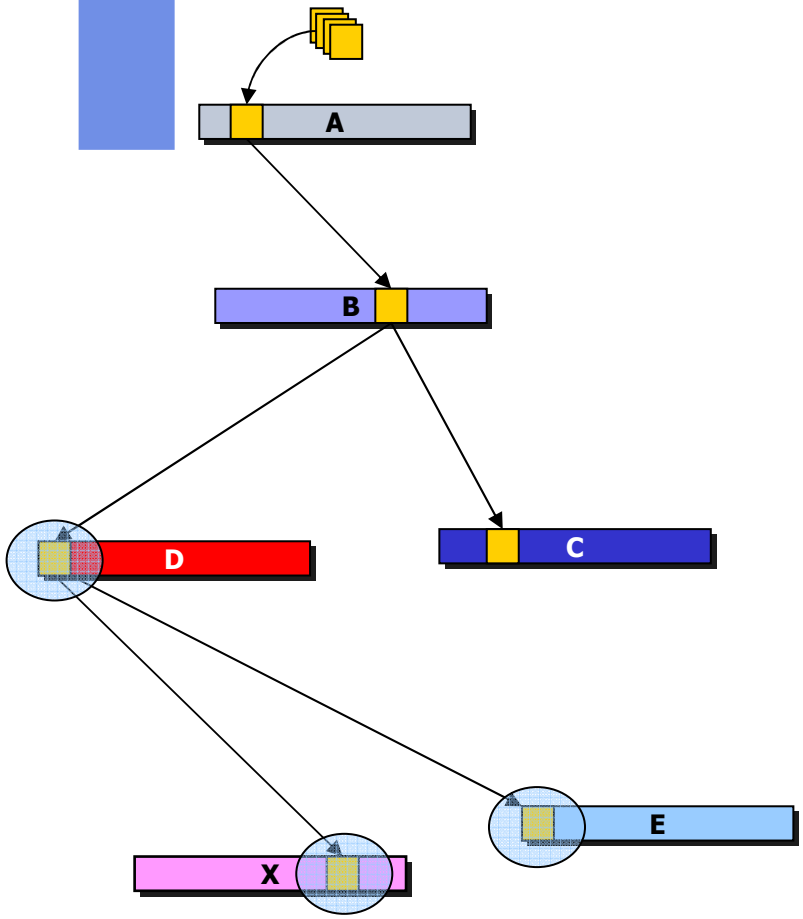
Mapping Database – Create Mapping

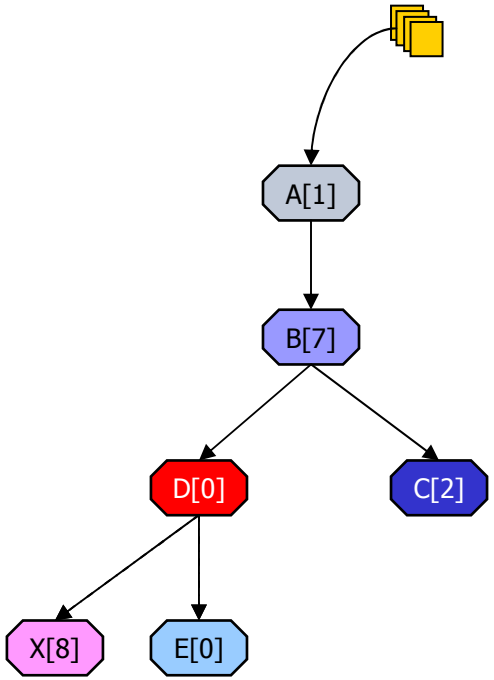


Mapping Database – Create Mapping

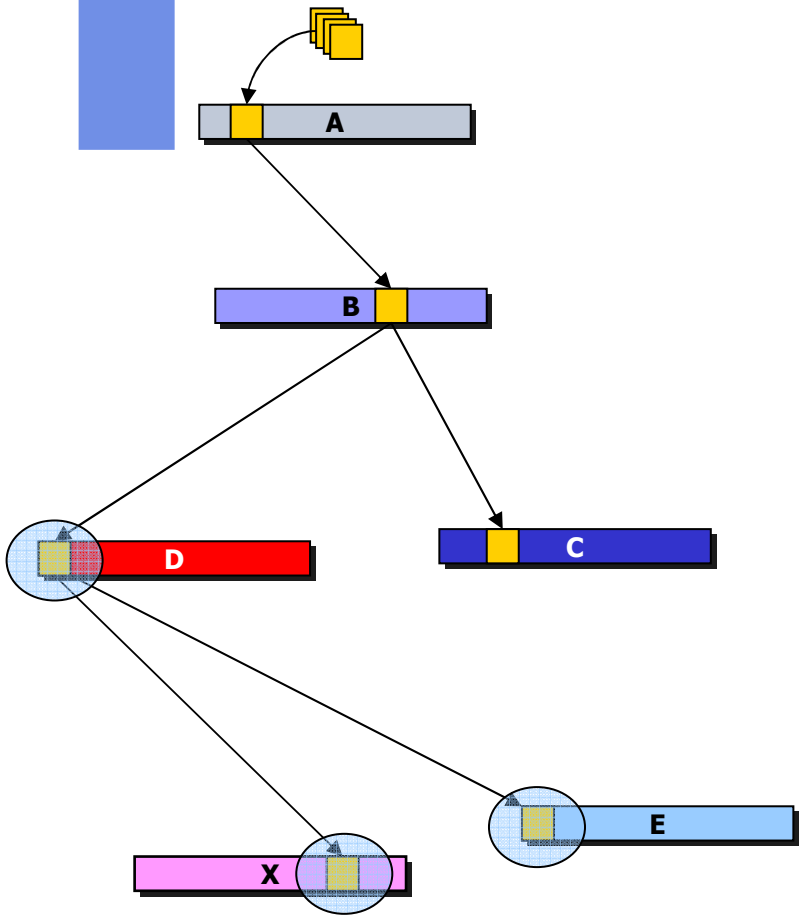


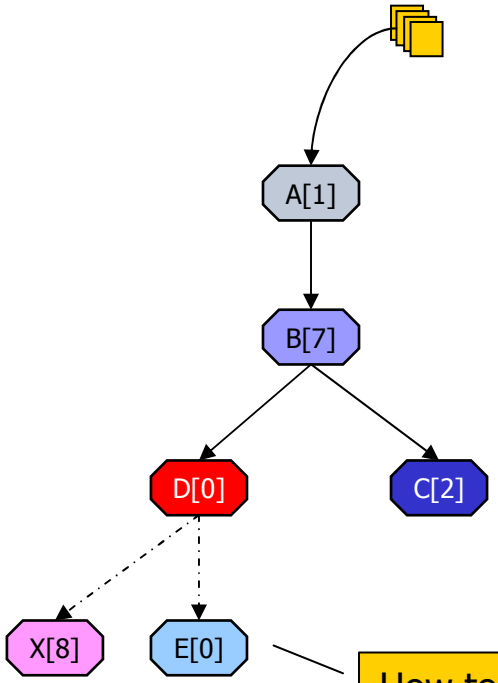
Mapping Database – Unmap





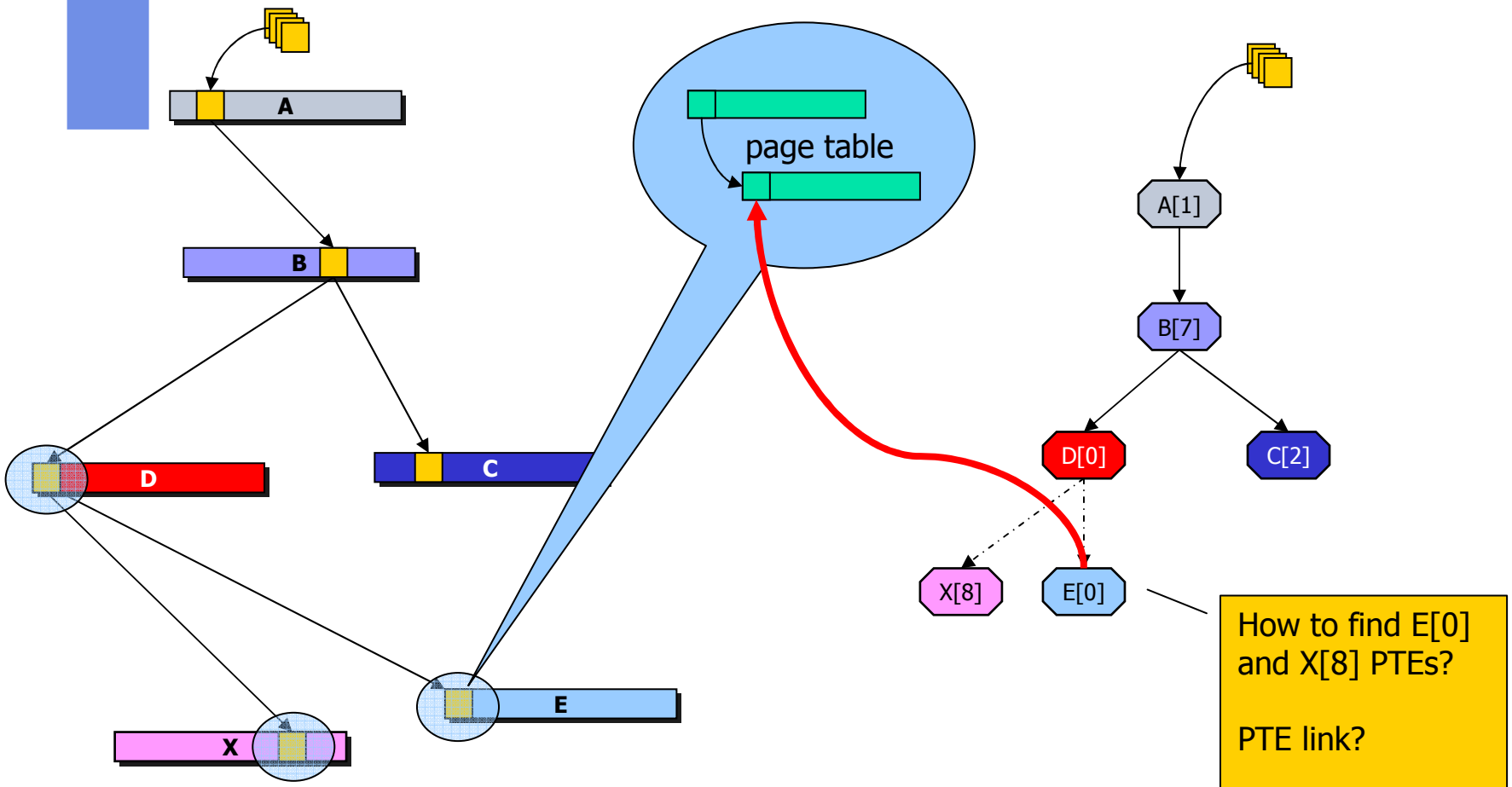
Mapping Database – Unmap



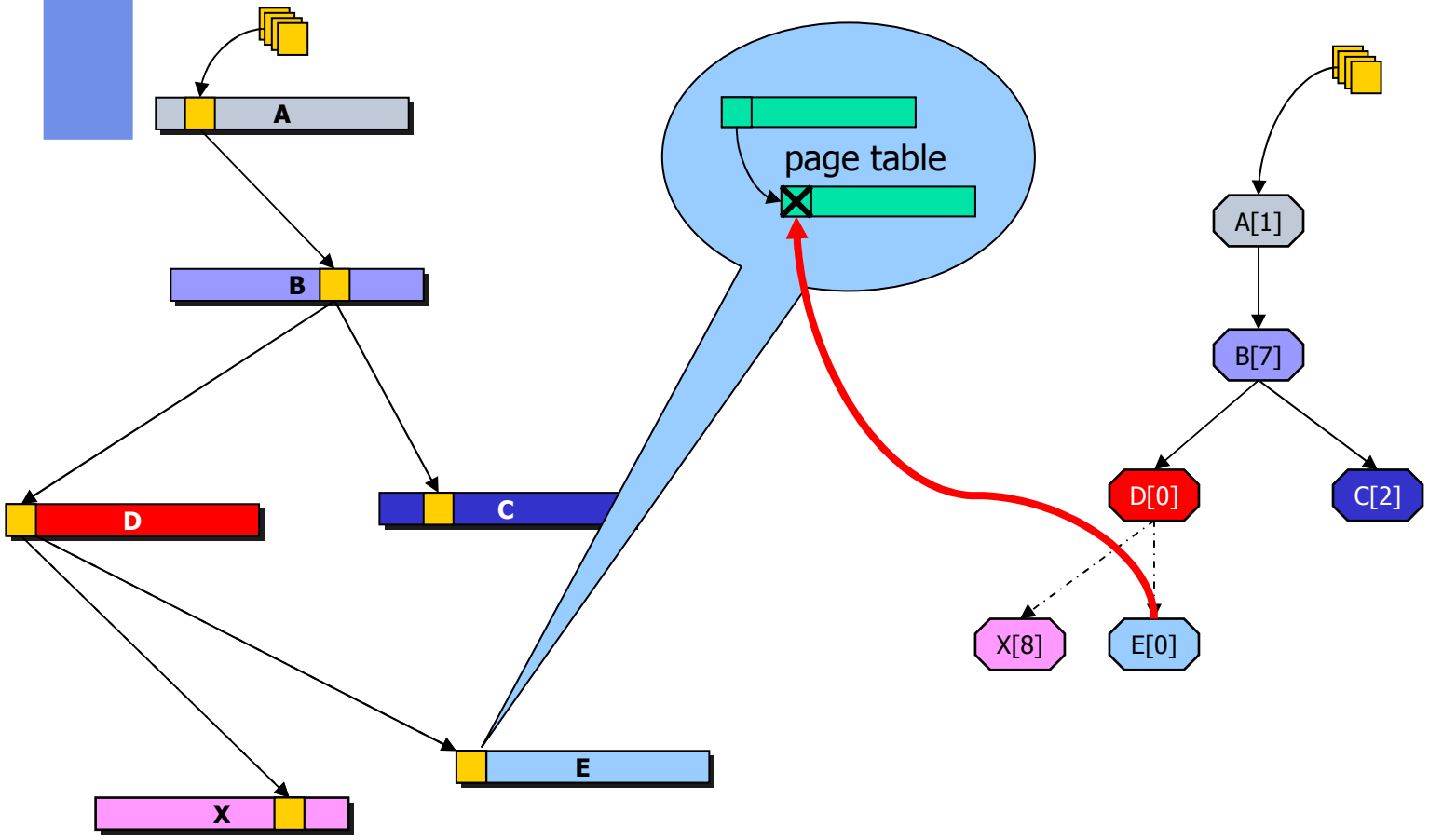


How to find E[0] and X[8] PTEs?

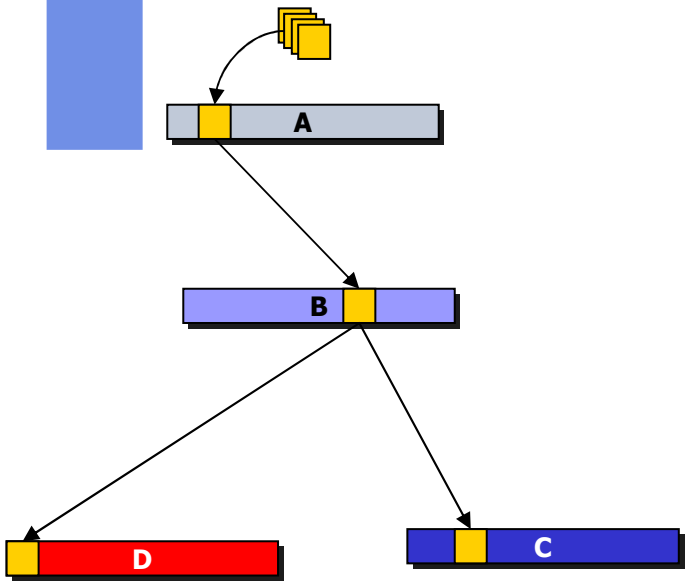
Mapping Database – Unmap

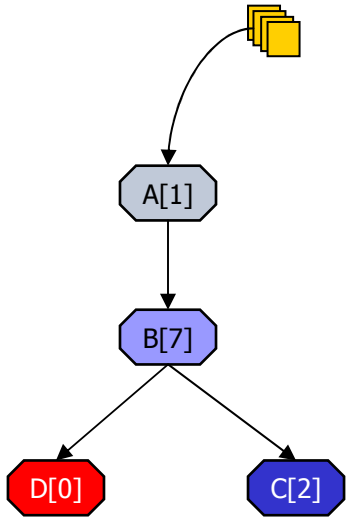


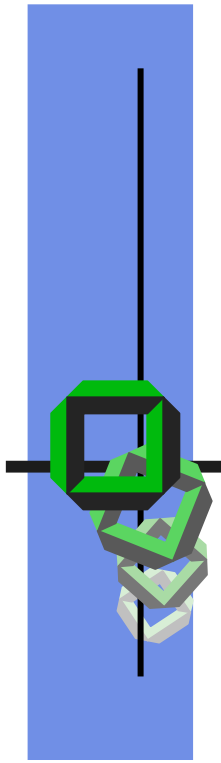
Mapping Database – Unmap



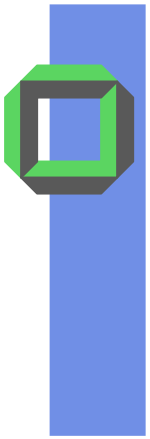
Mapping Database – Unmap







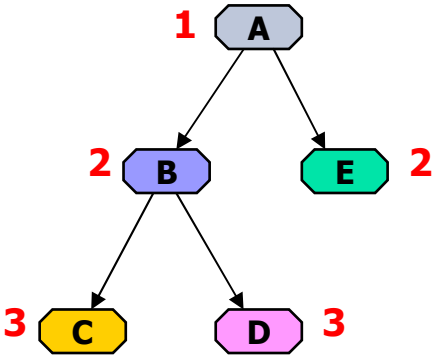
More Implementation

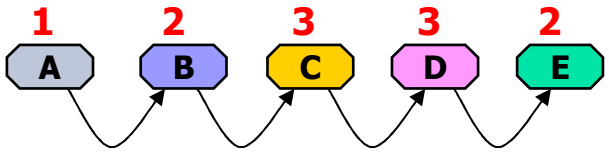


General Implementation Guidelines

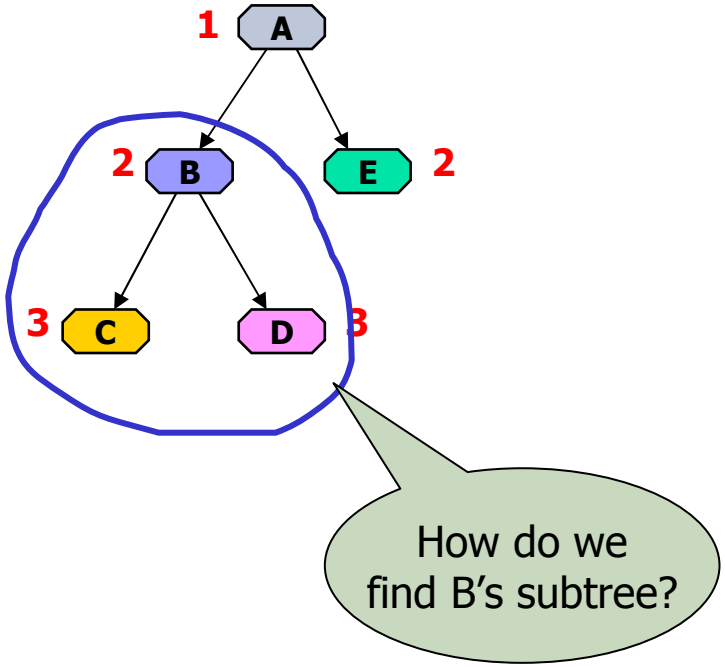
- Avoid recursion
  - Serialize recursive algorithms
- Allow for preemption
  - Implement appropriate locking mechanisms
- Minimize memory requirements
  - Make the right design decisions
  - Make structures compact
- Make it general

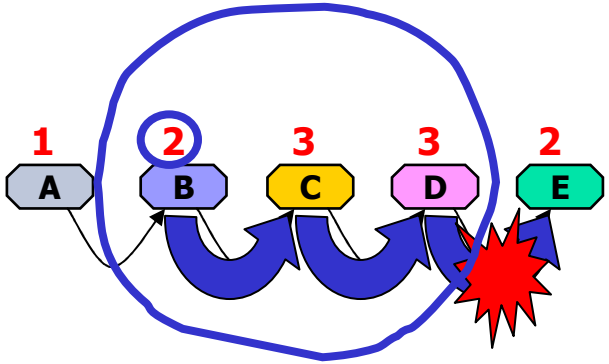
Avoiding Recursion – Basic Idea



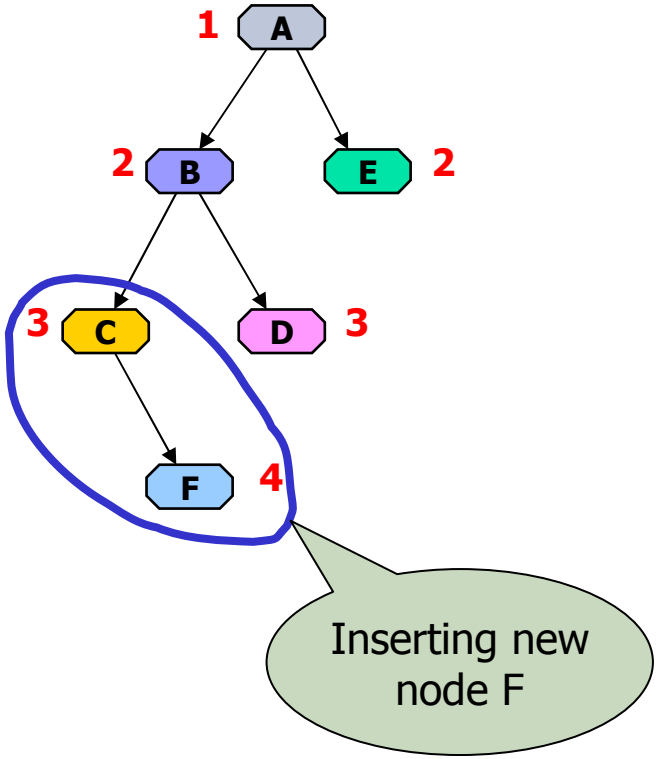


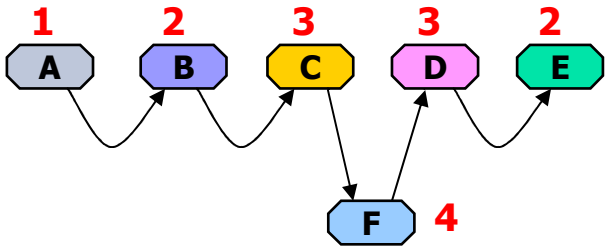
Avoiding Recursion – Parsing



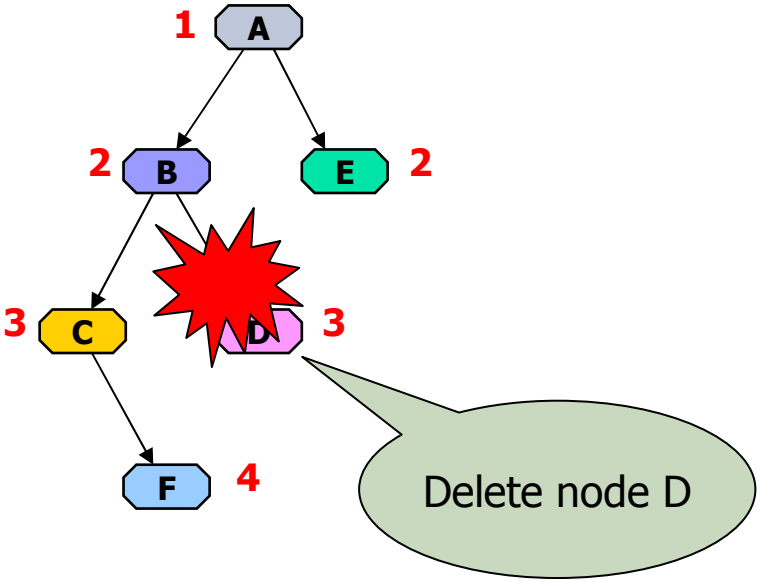


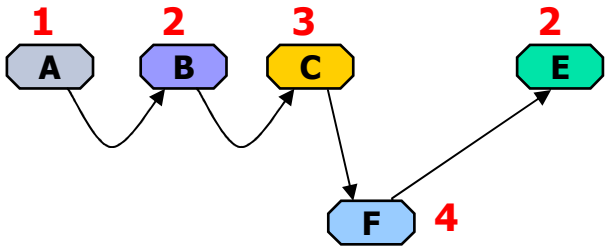
Avoiding Recursion – Insertion



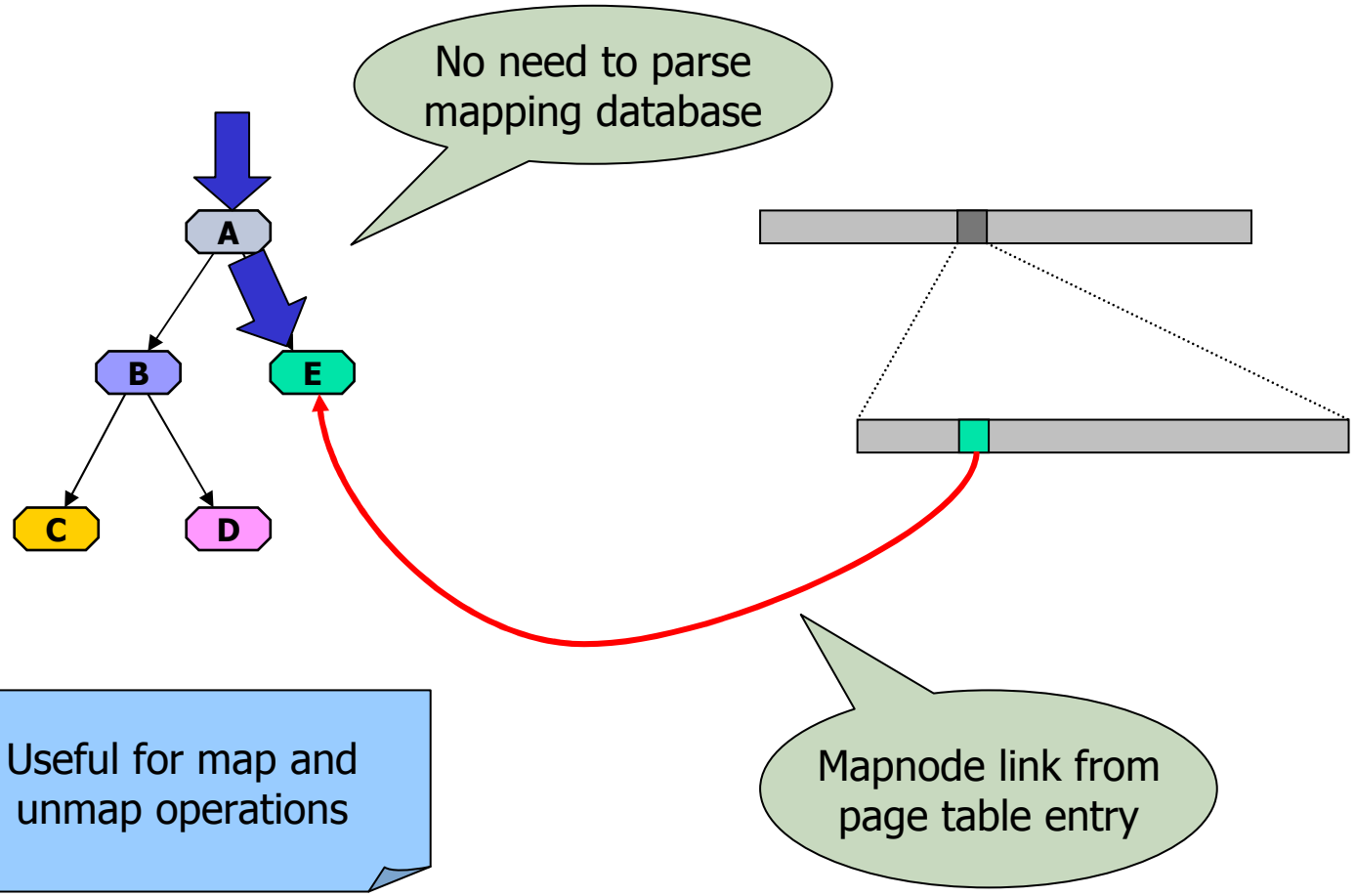


Avoiding Recursion – Deletion

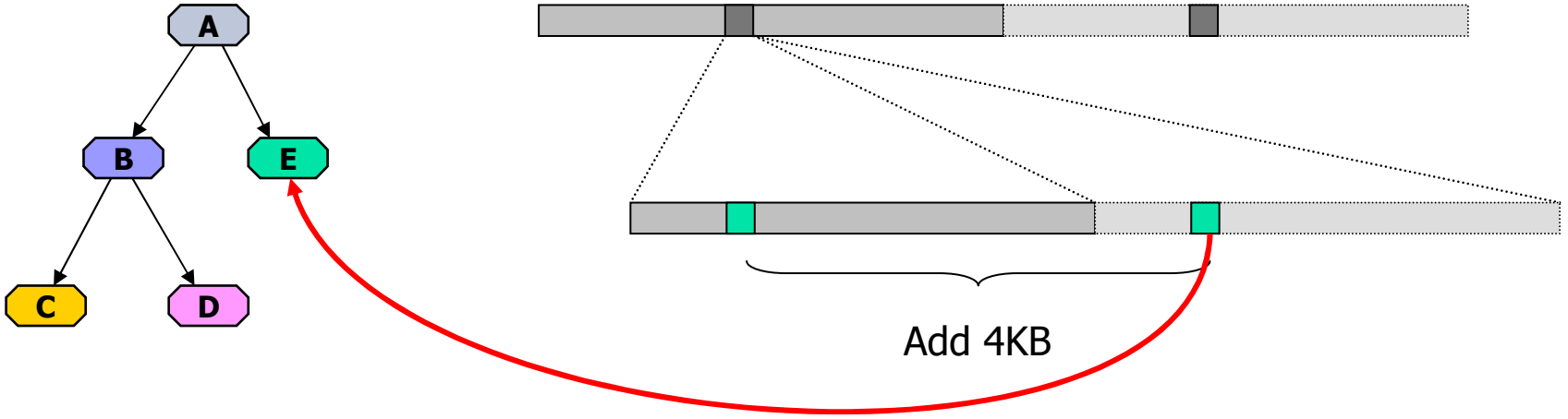




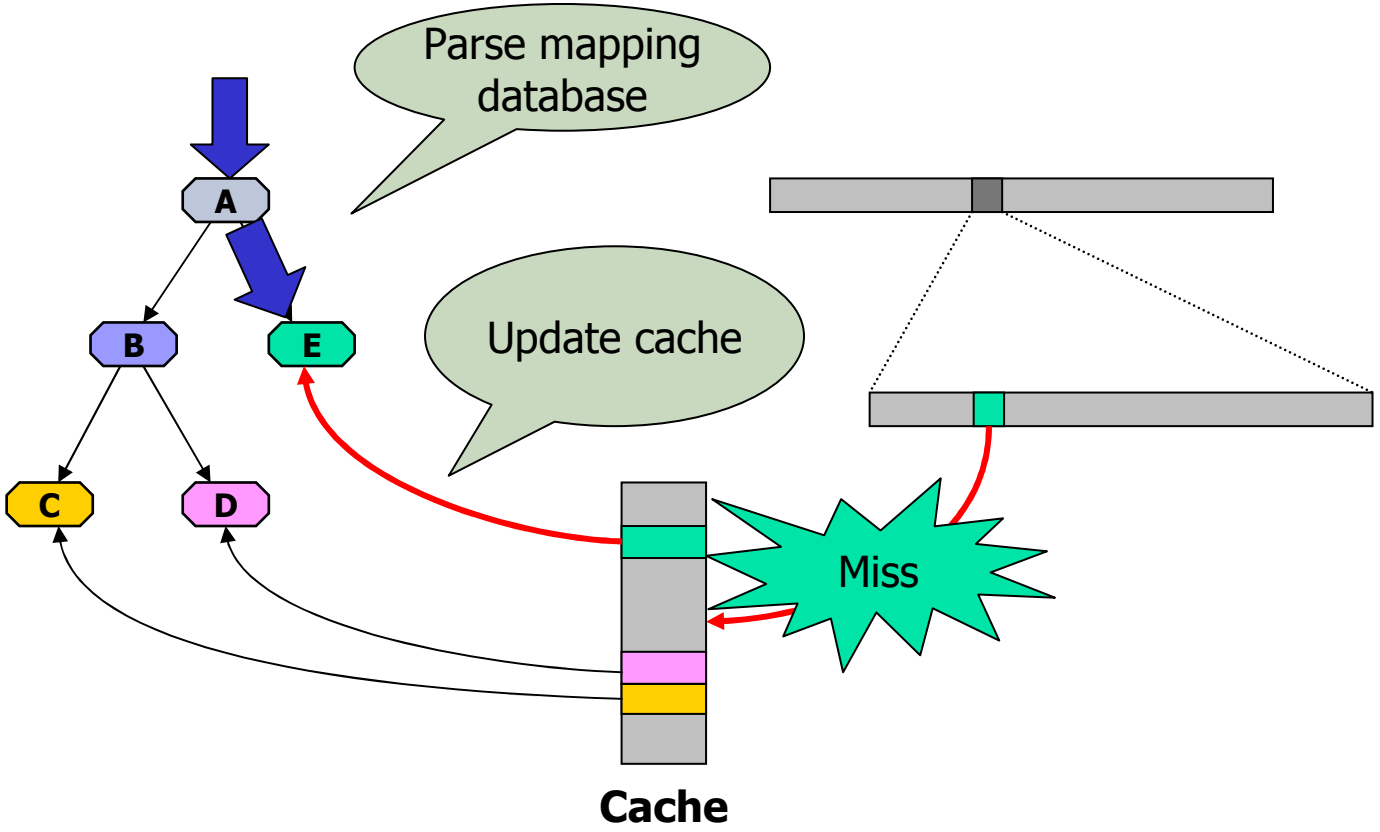
Design Decisions – Mapnode Pointers



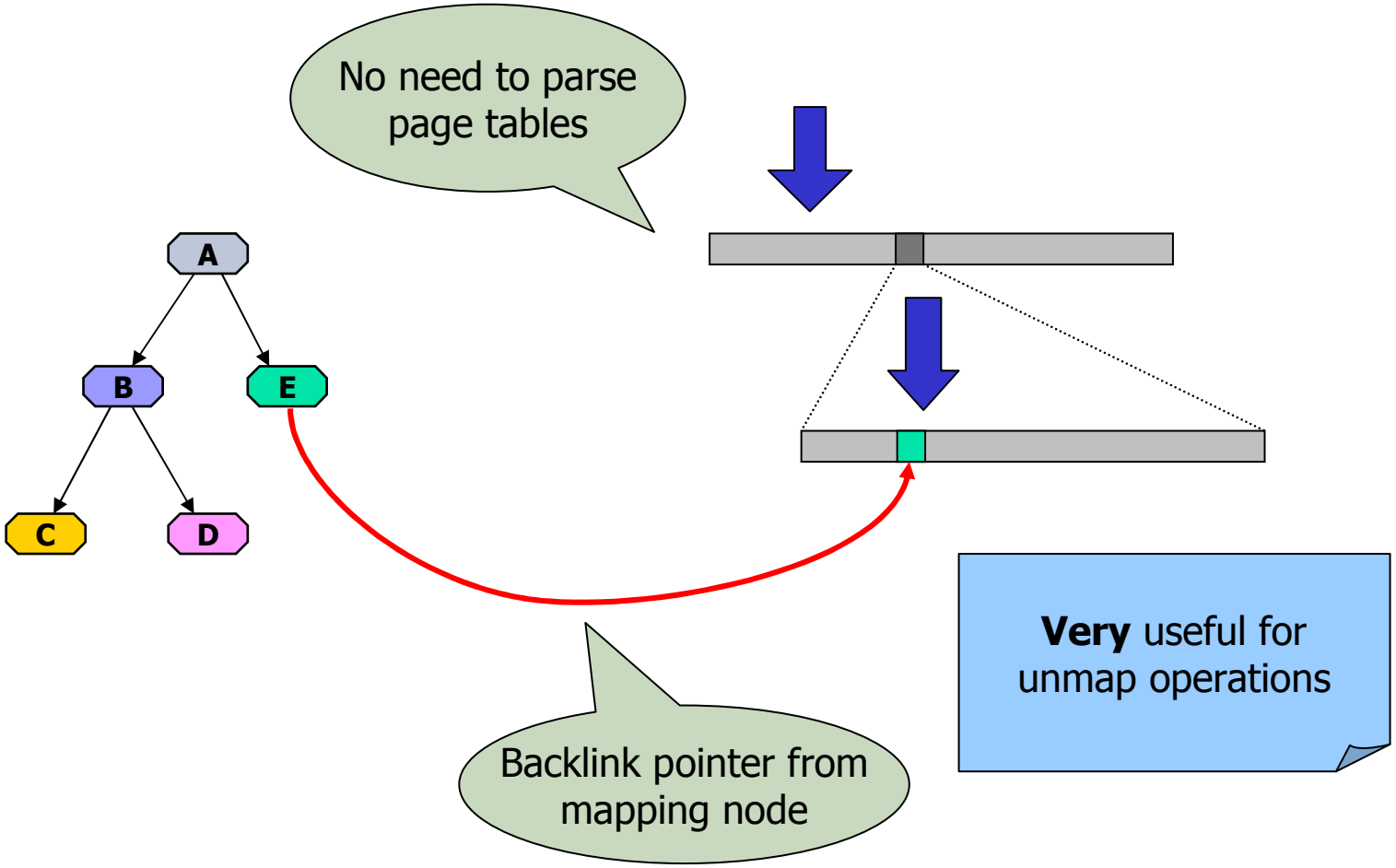
Design Decisions – x86 Implementation



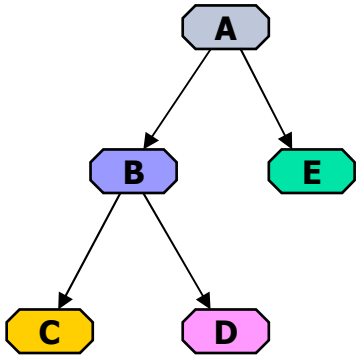
Design Decisions – Mapnode Caching

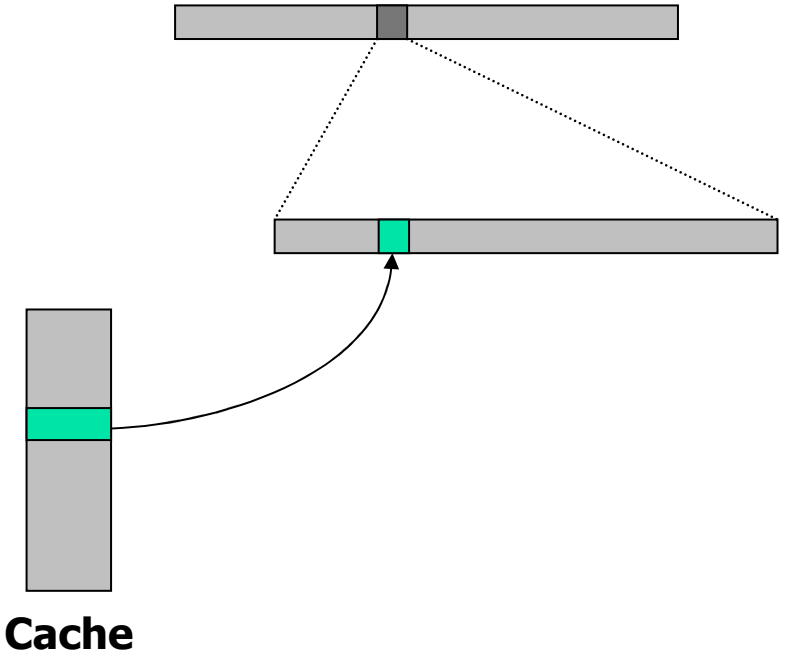


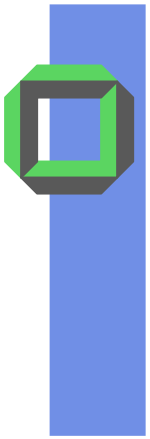
Design Decisions – Backlink Pointers



Design Decisions – Backlink Caching

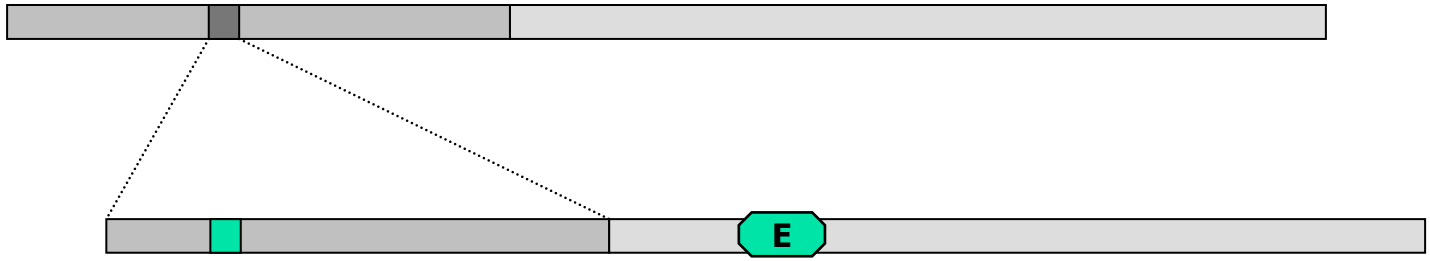


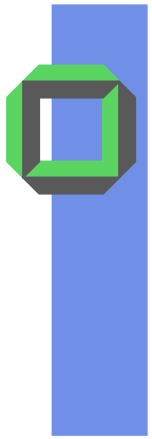




Design Decisions – Integrated Solution

- Page tables and mapping database are integrated
  - Simple with hardware independent page tables
  - A bit more tricky with hardware dependent page tables
  - Requires preallocation of all MDB nodes
    - At least for all entries in valid page tables
  - Don’t combine for cache reasons

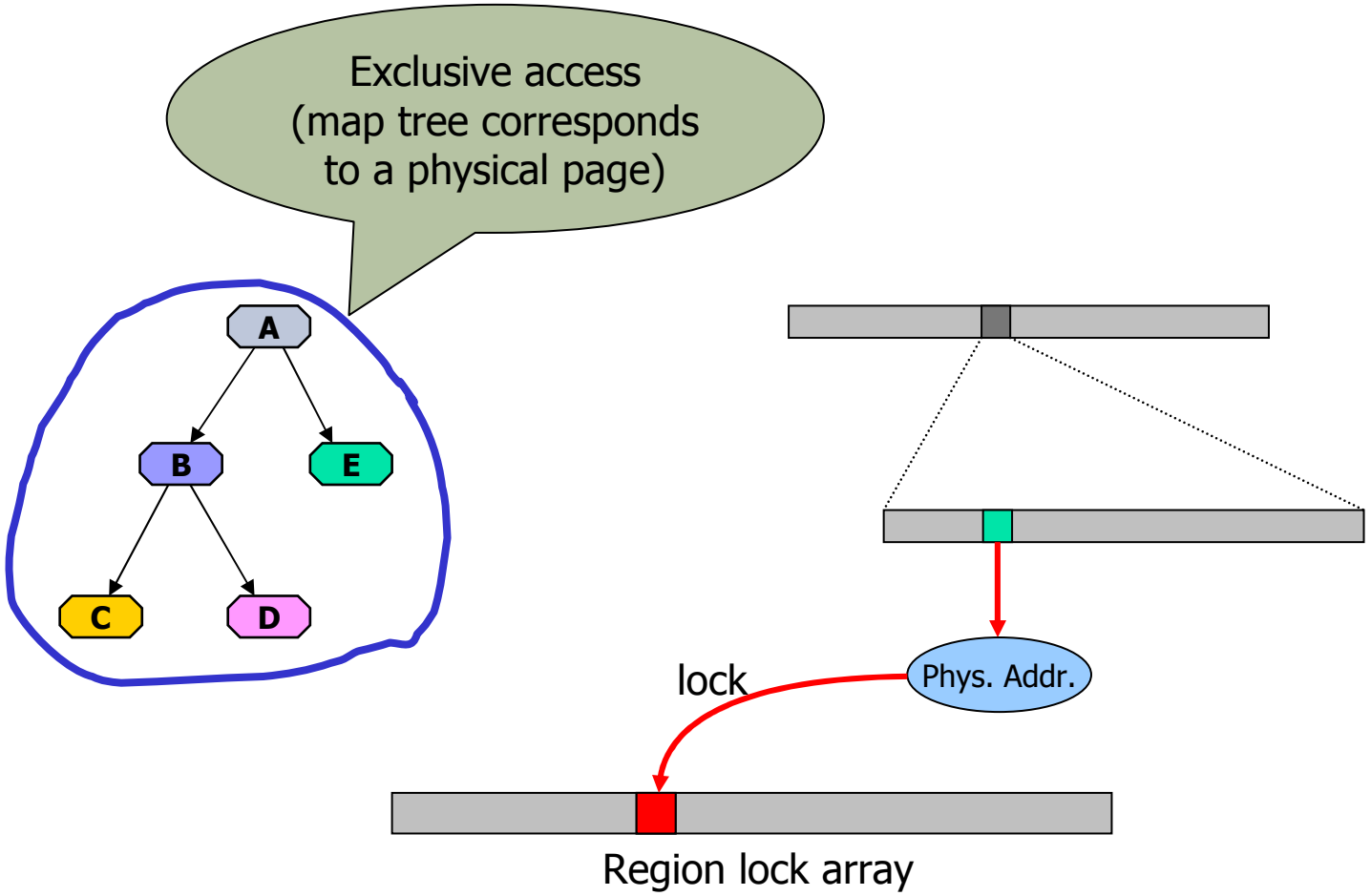


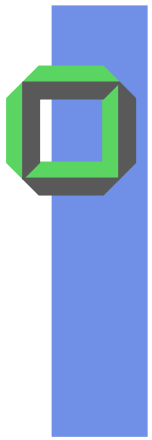


Locking

- Completely serialized
  - Only a single thread may map/unmap at a given time
- Coarse grained locking
  - Only a single thread may map/unmap pages inside some given physical region
- Fine grained locking
  - Only a single thread may access a given mapping node at a given time

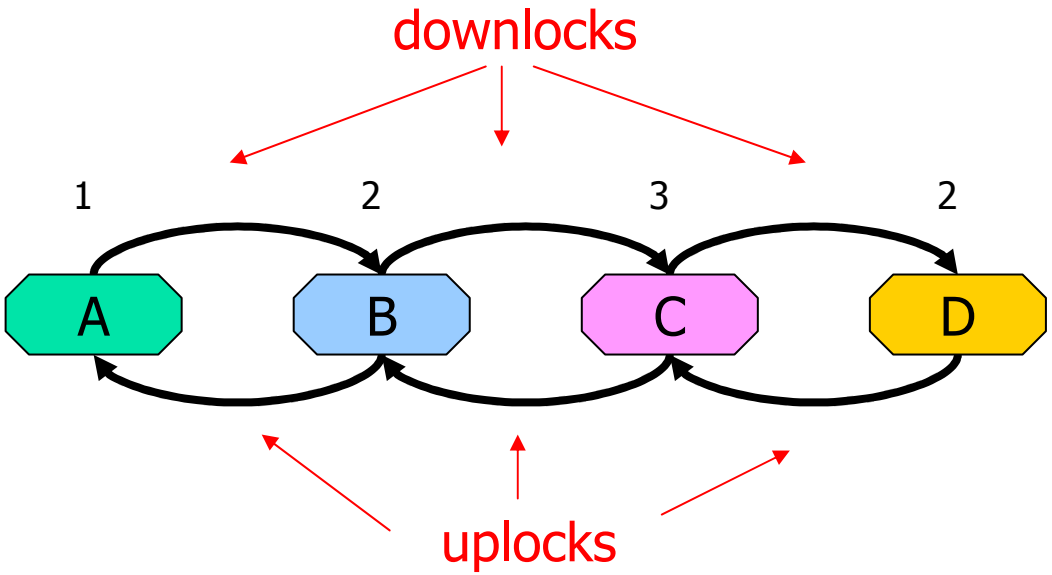
Locking – Coarse Grained

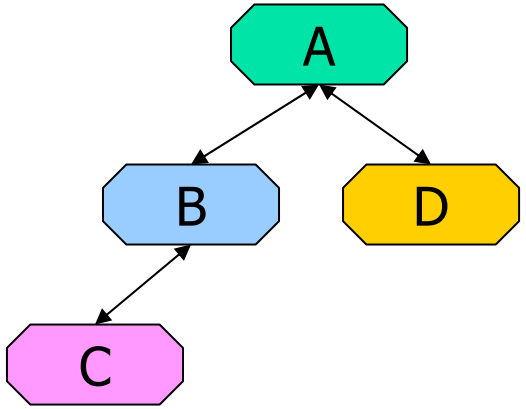


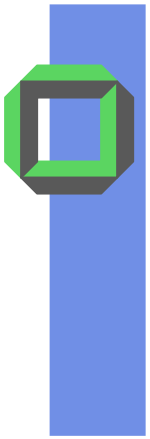


Locking – Fine Grained

- Each mapping node has two locks
  - Uplock protects previous (backlink) pointer
  - Downlock protects next pointer



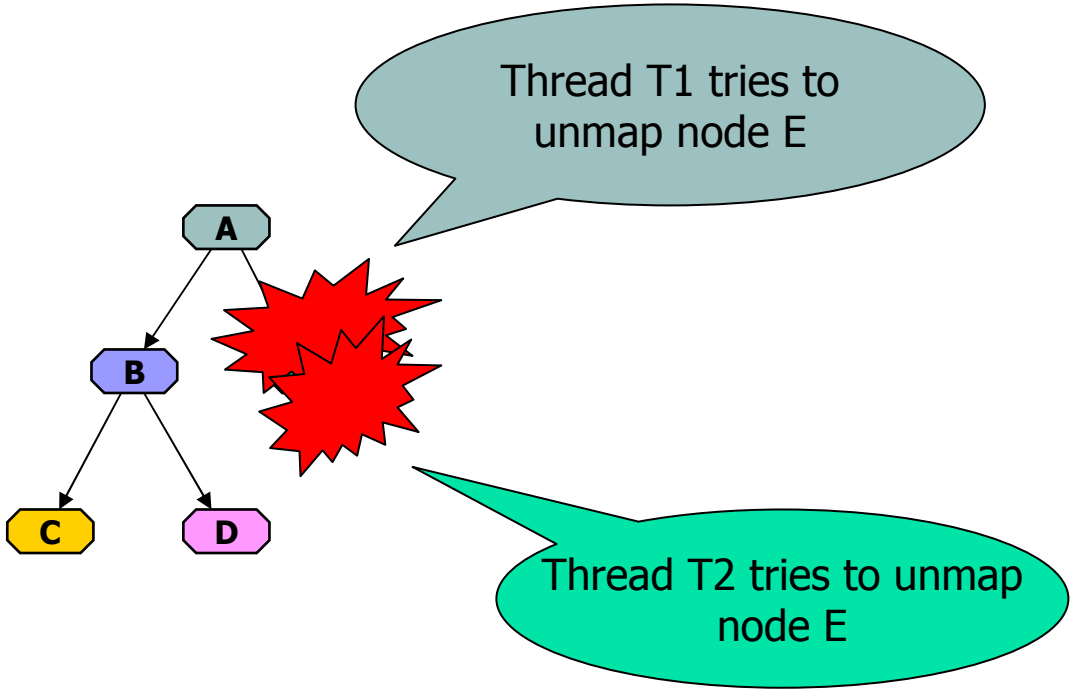




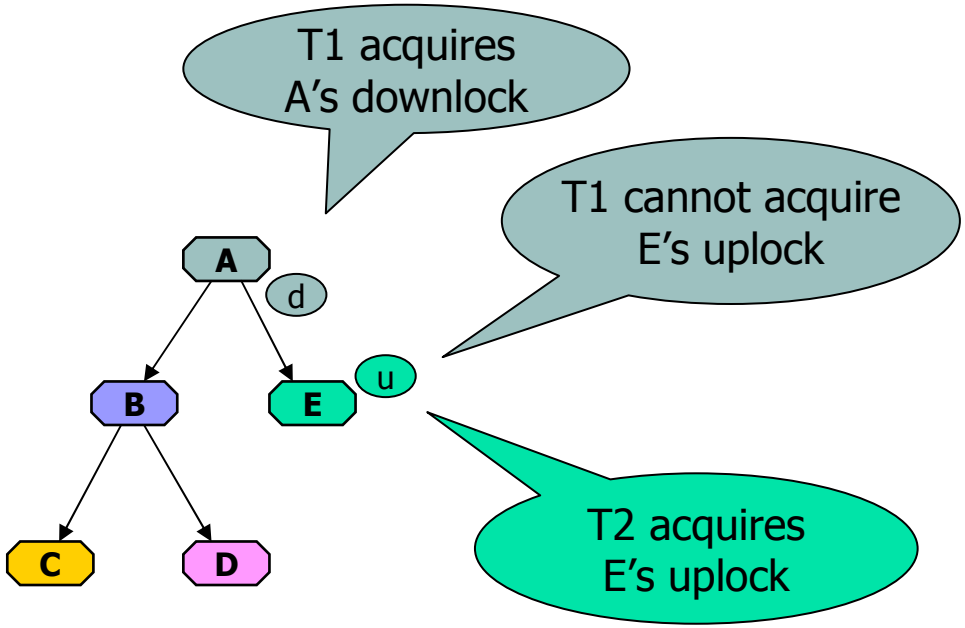
Locking – Fine Grained – Avoid Deadlock

- When an uplock is acquired
  - Thread may hold it as long as it wants
- When a downlock is acquired
  - Thread must release it if it does not manage to acquire the uplock of its child/sibling

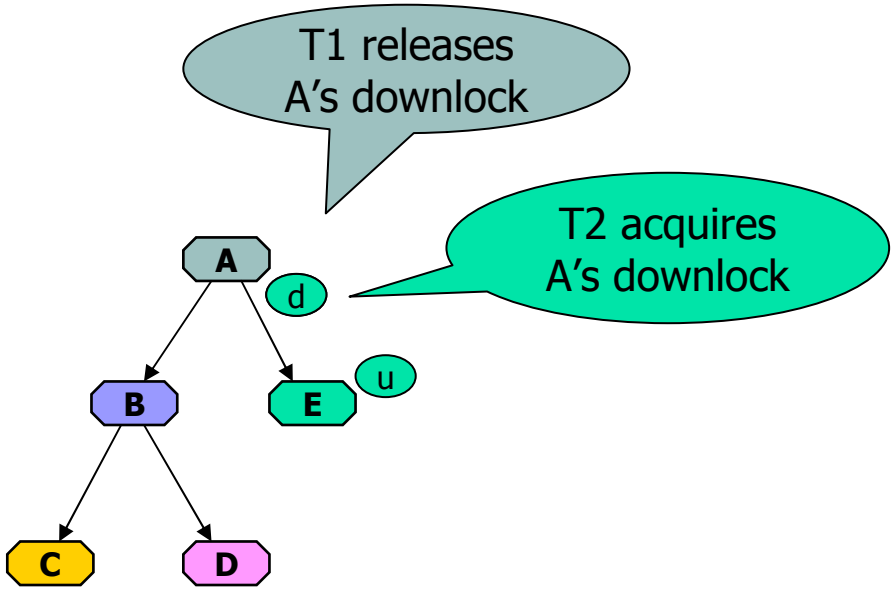
Locking – Fine Grained – Szenario 1



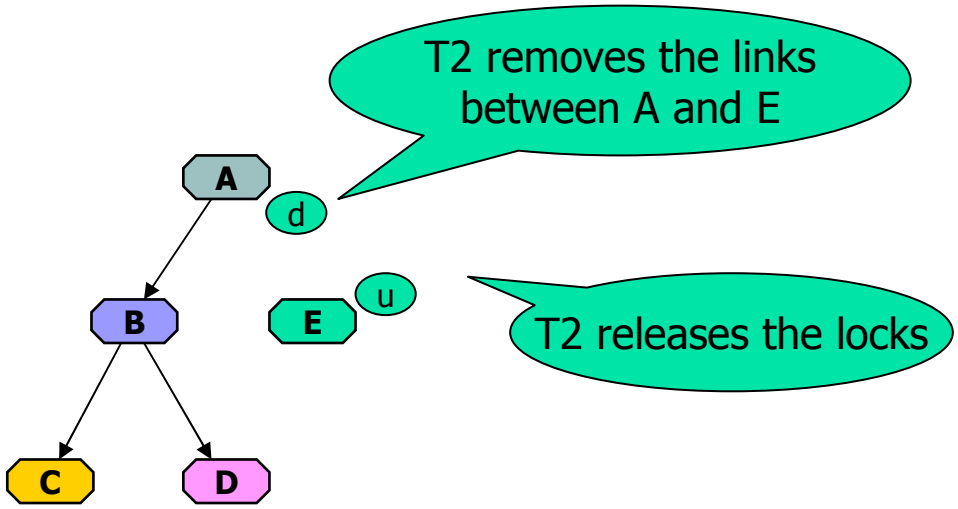
Locking – Fine Grained – Szenario 1



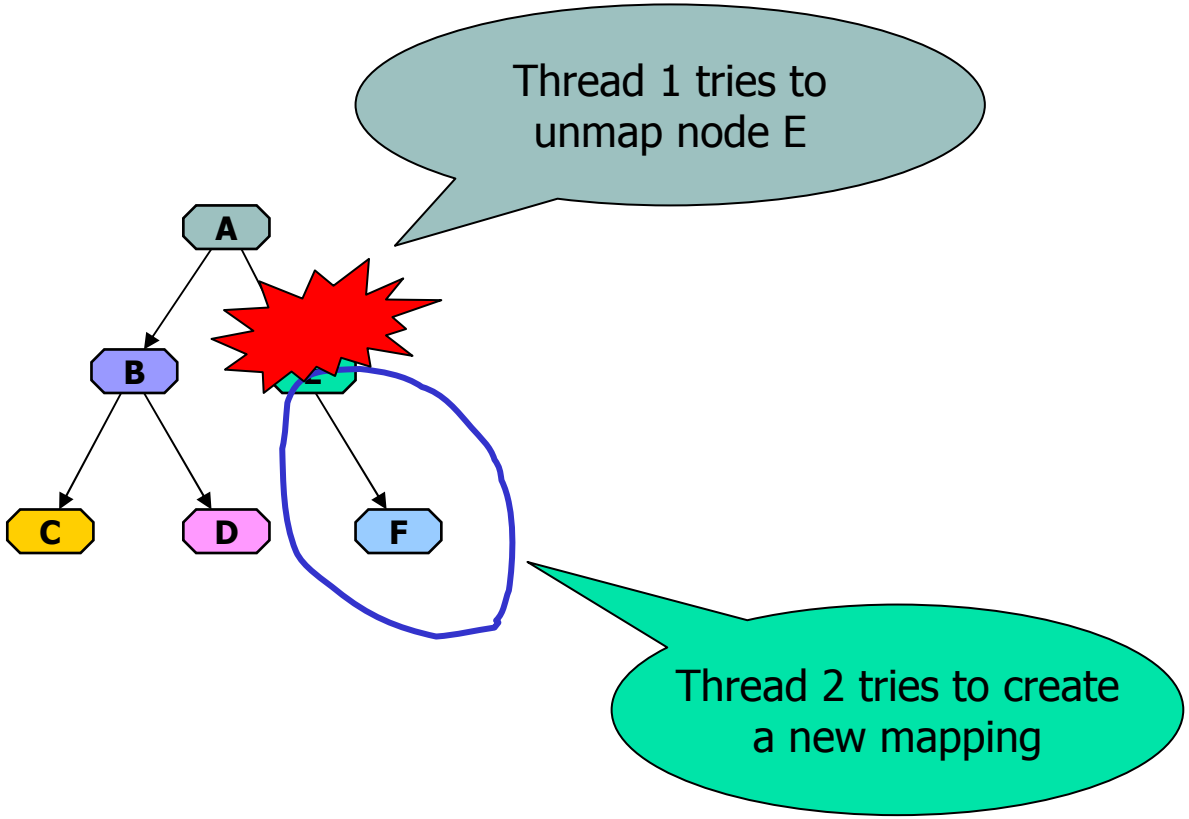
Locking – Fine Grained – Szenario 1



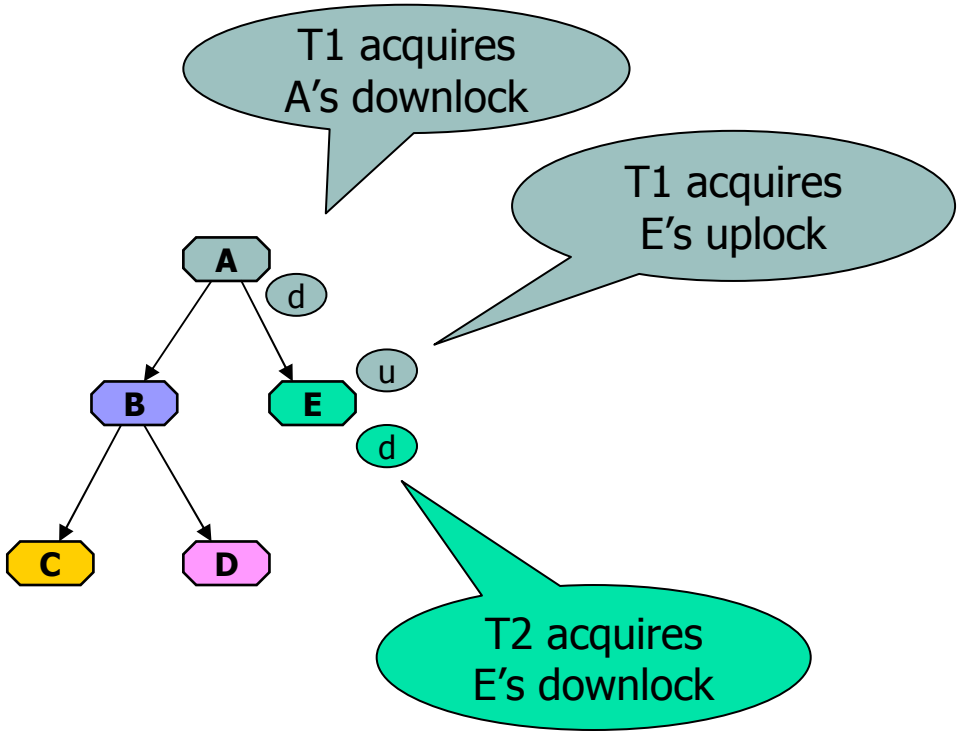
Locking – Fine Grained – Szenario 1



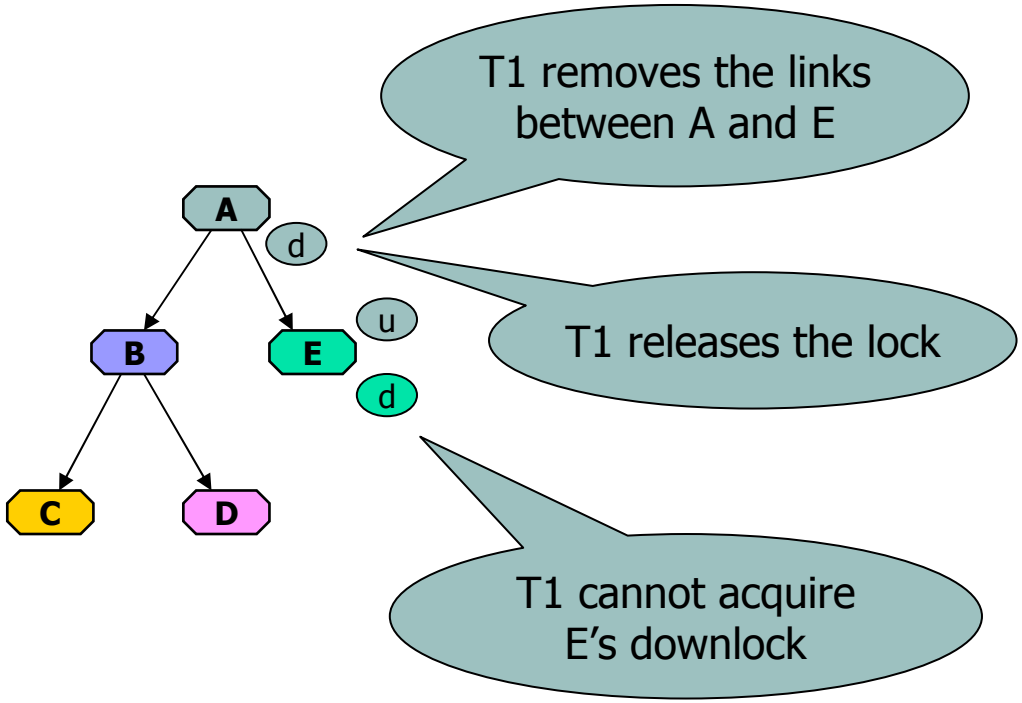
Locking – Fine Grained – Szenario 2



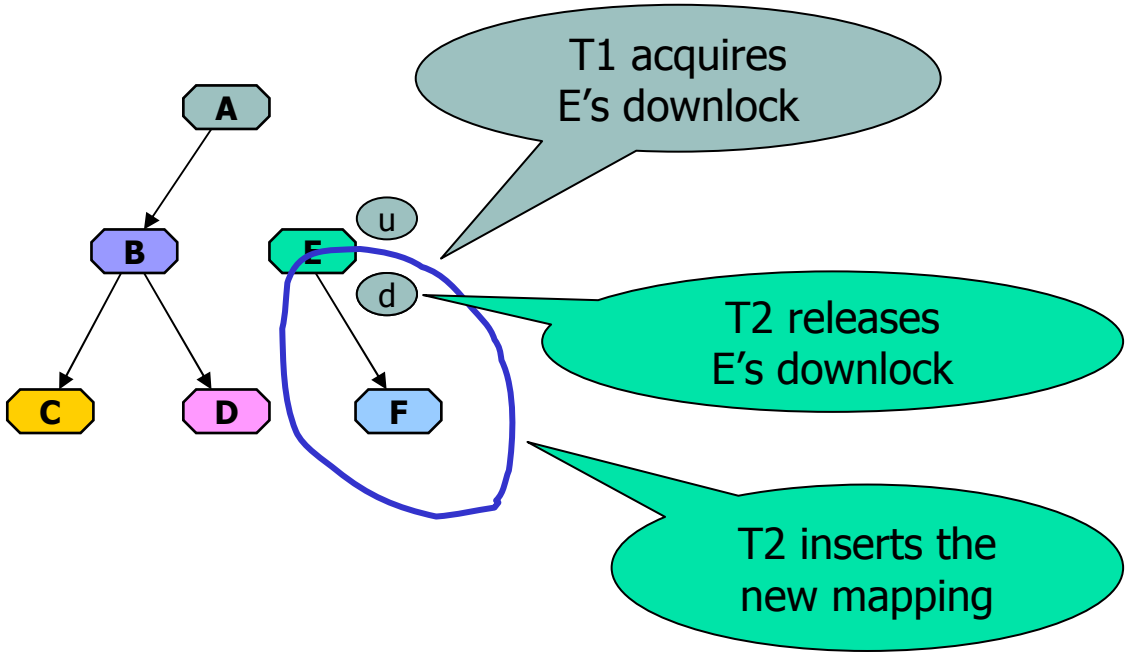
Locking – Fine Grained – Szenario 2

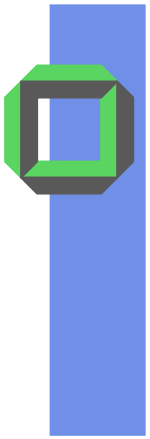


Locking – Fine Grained – Szenario 2



Locking – Fine Grained – Szenario 2





Structure Packing

- Use only required number of bits
  - E.g., page directory address does not require 32 bits
- Store multiple values in one word
  - Occasionally one of two values is known
    - When iterating a doubly linked list, either the previous (or the following) node is known
    - Must iterate from start or end (cannot start in between)
  - Store only ‘`valueA XOR valueB`’
  - Then
    - `valueA == stored value XOR valueB`
    - `valueB == stored value XOR valueA`