Deadlocks Basics

P. J. Denning

For CS471/CS571

Definition

- System has many concurrent tasks (processes, threads).
- Tasks require resources to make progress.
- System has fixed pools of resources of different types (CPU, pages, sectors, files, records, etc.).
- Each task holds some resources
- Each task will have to wait if it requests more resources that are not available.
- Deadlock = circular wait among tasks.

Two kinds of deadlock

- Signals (consumable resources)
  - Each task waiting for signal from another
  - Can't back out
- Units (reusable resources)
  - Each task waiting for another to release a resource unit
  - Can back out by aborting tasks until enough resources available to free others.

Signal Deadlock

```
P1: ...
    wait(a)
    wait(b)
    use objects a and b
    signal(b)
    signal(a)
    ...
```

```
P2: ...
    wait(b)
    wait(a)
    use objects a and b
    signal(a)
    signal(b)
    ...
```

Initially, semaphores a and b are both 1.
Deadlock if P1 and P2 complete their
first wait's simultaneously before
attempting their second wait's.

May not be easy to test for signal deadlocks:

```
T:  transaction(a,b)
    wait(a)
    wait(b)
    use objects a and b
    signal(b)
    signal(a)
    return
```

Deadlock may result if P1 calls T(a,b) and
P2 calls T(b,a) at the same time.

Two Phase Locking

Database systems use two-phase protocol:
get all locks before updating; don't wait.

Operation lock(r) returns lock value and sets lock

```
T:  transaction(a,b,c)
    if lock(a) then goto T
    if lock(b) then {unlock(a); goto T}
    if lock(c) then {unlock(a,b); goto T}
    update records a,b,c
    unlock(a,b,c)
    return
```

Two-phase protocol may have long busy-waiting period during times of contention for shared records.

Can reduce waiting probability by inserting random delay before looping back after finding locked records.

Resource Deadlock

- Circular waits arising from waiting for new resources to be granted while holding other resources.
- Can be "backed out of" by aborting one or more of the deadlock processes and releasing their resources back to the system pools.

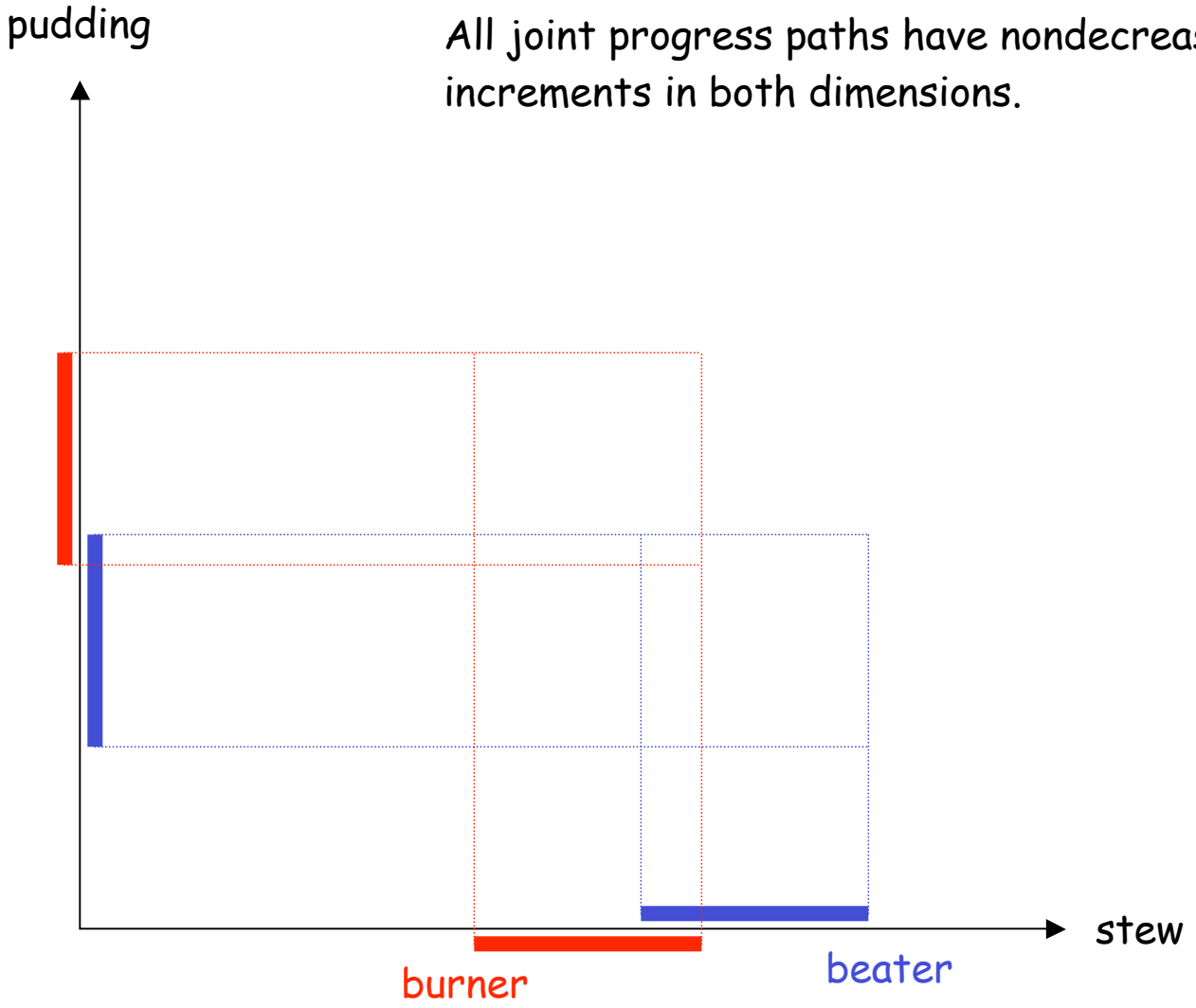
Banking Example

- Bank gives Alice credit limit $100 and Bob $200. Loan pool is $250.
- Alice asks for $90, bank grants. (Pool = $160)
- Bob asks for $160, bank grants. (Pool = $0)
- Alice asks for $10, waits.
- Bob asks for $10, waits.

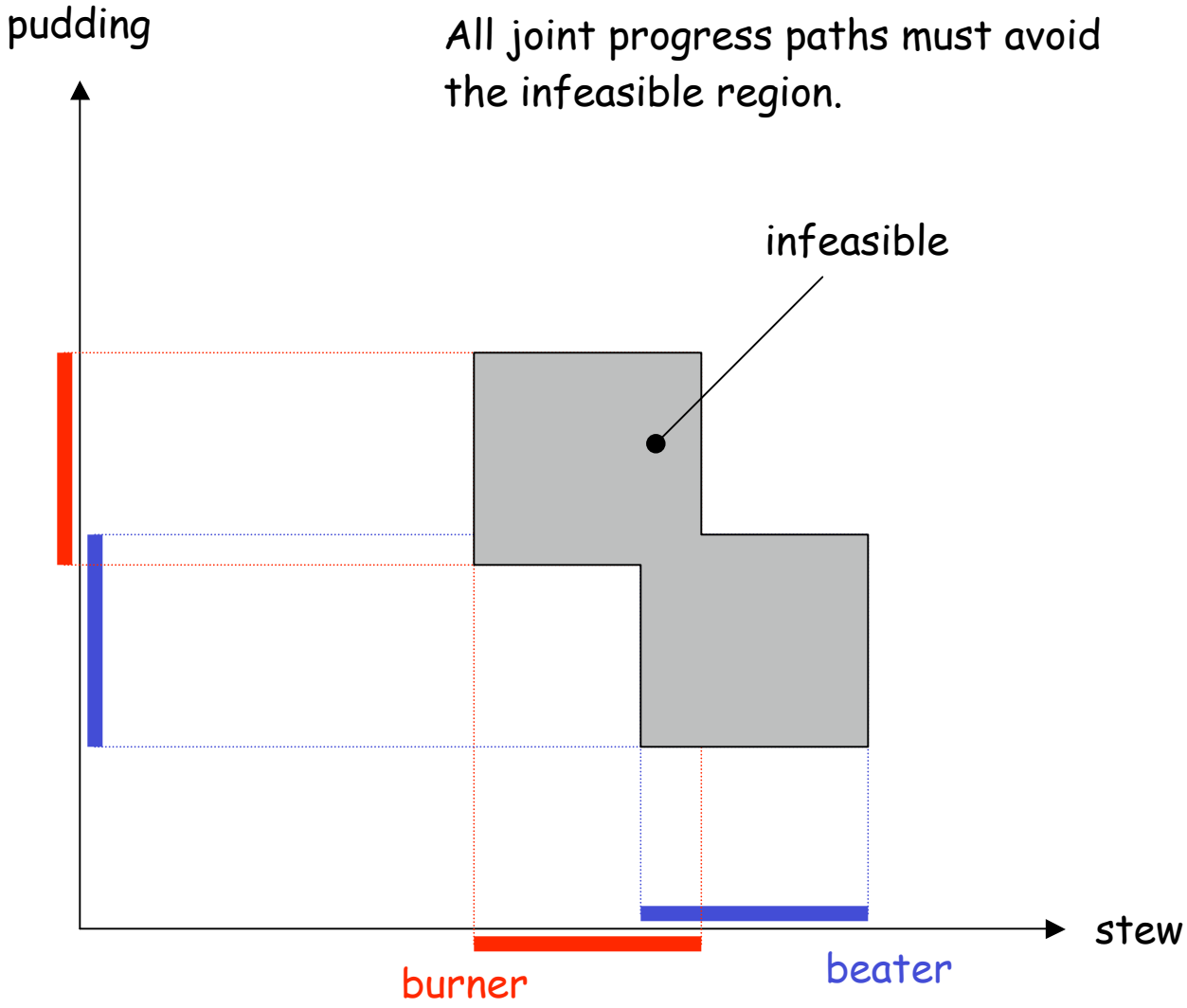
Kitchen Example

- Kitchen has two resources, burner and beater.
- Chef has two recipes, stew and pudding.
- Stew: start beating while still cooking on burner, continue beating for a few minutes after removing from burner.
- Pudding: start beating before placing on burner, continue beating for a few minutes after placing on burner.

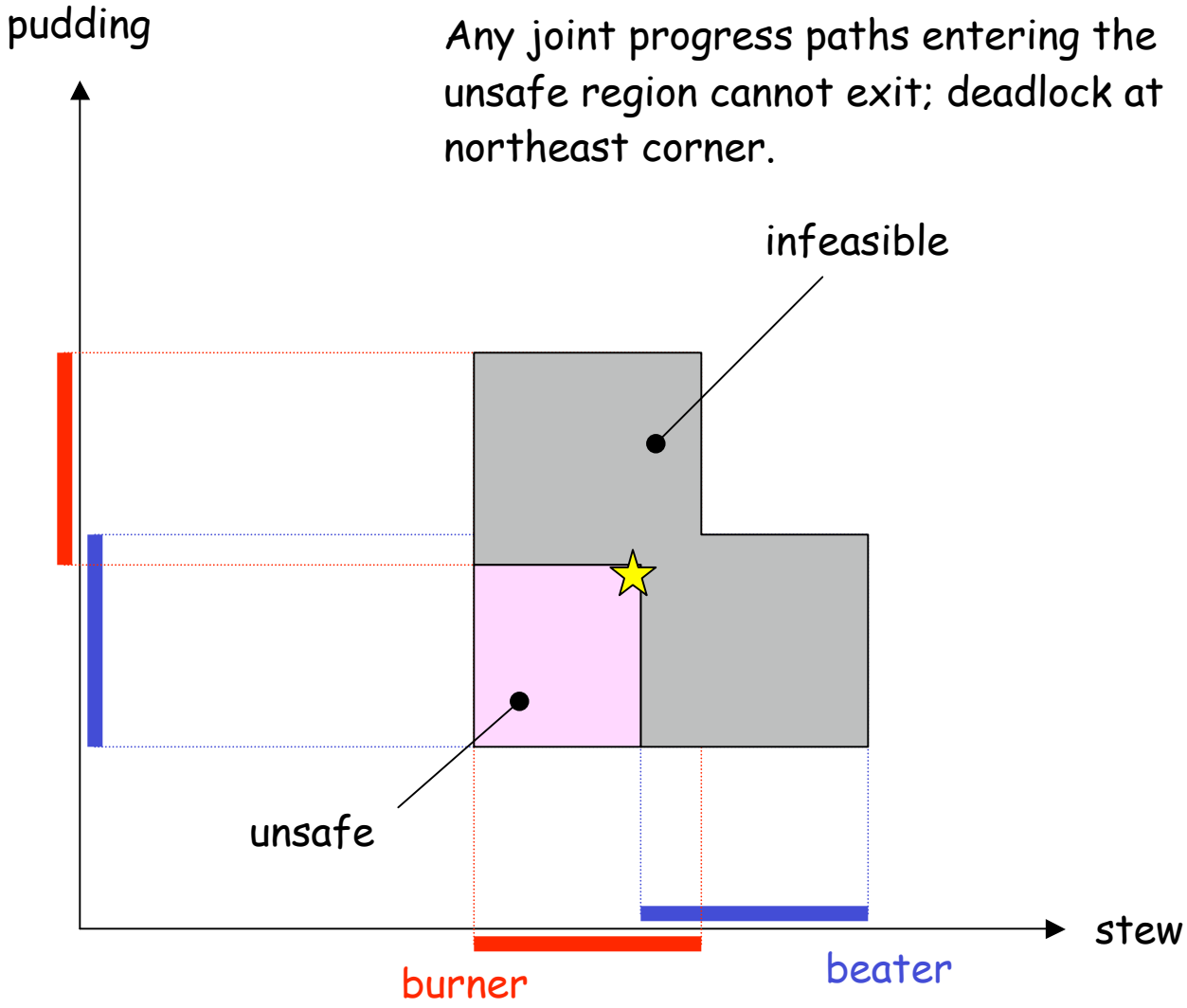
All joint progress paths have nondecreasing increments in both dimensions.



All joint progress paths must avoid the infeasible region.



Any joint progress paths entering the unsafe region cannot exit; deadlock at northeast corner.



Memory Example

- Command SEND(p,m) places message m in a buffer obtained from OS and attaches to inbox of process p.
- Command m=GET() returns the first message from inbox and returns buffer to OS.
- Deadlock can result if OS runs out of buffers when p1 and p2 are attempting SEND to each other at the same time.

Deadlock condition

For each deadlocked task, the set of unfilled requests is not covered by the sum of the available resources plus all resources held by non-deadlocked tasks.

Phrasing works for multi-dimensional resources where "A covers B" means that each dimension of A is greater than or equal to the corresponding dimension of B.

Draw graph with nodes for tasks and resources; arrow (T,R) means task T waits for resource R; arrow (R,T) means T holds resource R. Deadlock implies loop in graph. However, loop does not imply deadlock.

Basic Deadlock Detection Algorithm

Set D = {all tasks}

Set A = vector of resources available

Find i in D such that A covers request(i);
    remove i from D and
    add holdings(i) to A

Repeat until no more i's.

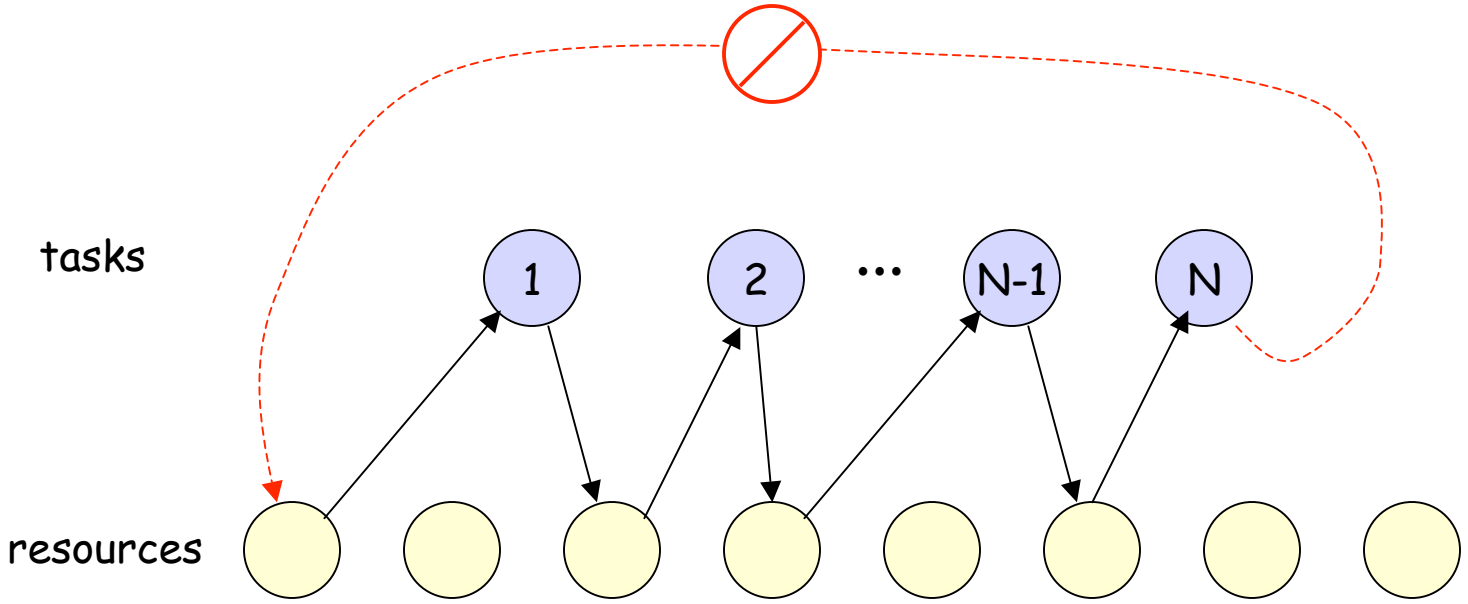
Then D is the set of deadlocked tasks.

Deadlock Avoidance

- prior prevention -- negate one of the essential conditions
  - mutual exclusion of units
  - nonpreemption of units
  - resource waiting
- detection and recovery
- dynamic control of joint progress paths

Prevention

- grant all resources before starting task
  - two-phase locking
  - max at beginning
- ordered resource usage:
  - resources in groups 1,2,3,...
  - if task requests more, they must come from higher numbered group than any in which task holds resources.



If tasks 1,2,...,N are in circular wait, then task N must be holding the resource needed by task N-1 and requesting the resource held by task 1. This is impossible under ordered usage.

- Ordered resource usage can be applied to the locking protocol for database records.
- Sort the list of requested records in order (e.g., by id) and request in that order.

- Prevention is conservative: it withholds resources that may not be used.
- Tradeoff between unused resources and time otherwise spent detecting and removing deadlocks.

Detection

- If deadlock suspected, apply "reduction" algorithm: pretend all tasks deadlocked; find task that can complete from available resources; delete it from the set and pretend its resources are released and added to those available; repeat.
- If algorithm stops with nonempty set of tasks, they are the deadlocked ones.

Path Monitoring

- Generally intractable because path can enter unsafe region, in which progress is still possible but deadlock inevitable; finding a safe path is exhaustive search problem.
- One tractable case: each task has (multi-dimensional) line of credit; granting a task's next request is safe if the next state, modified by pretending all tasks reach their credit limits, contains no deadlock.