Dynamic Power Management (DPM)

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

14, Jan., 2015

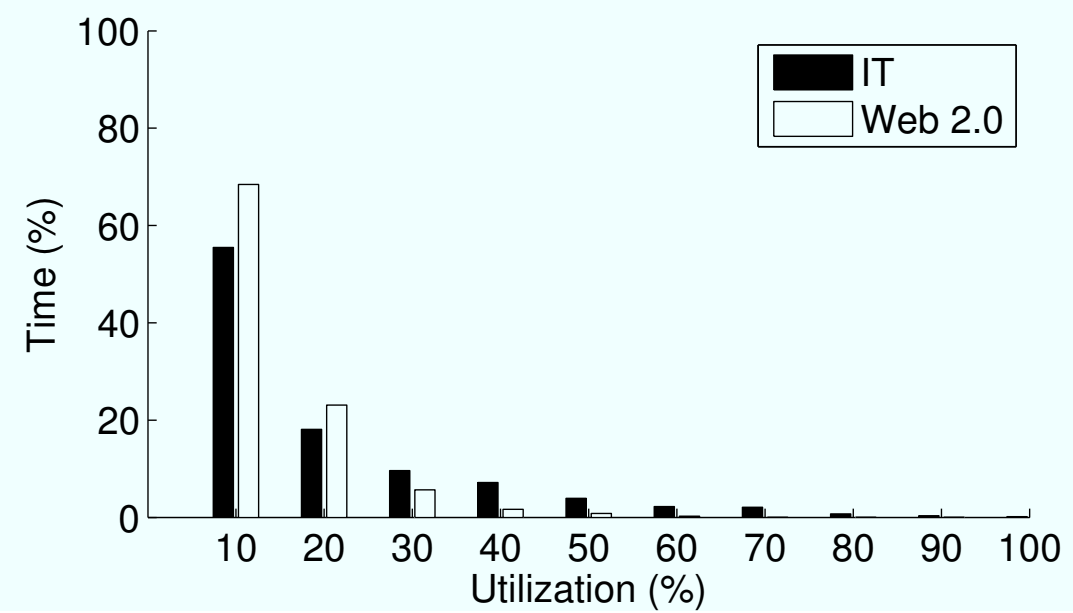
Today's Topic

Problem Definition:

- Given a set of jobs with known arrival times, deadlines, and required execution time
- How do we schedule when to turn on and turn off the system so that the jobs meet their timing constraints and the energy consumption wasted for turning on/off is minimized?

- Philippe Baptiste, Marek Chrobak, Christoph Dürr: Polynomial Time Algorithms for Minimum Energy Scheduling. ESA 2007: 136-150
- Philippe Baptiste: Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. SODA 2006: 364-367

When to Sleep or Turn Off Matters



The average server utilization is only 20-30% in typical servers.

Problem Definition

- Input: a set $\mathcal{J}$ of n jobs, in which job $J_j \in \mathcal{J}$ is with
  - arrival time a_j
  - absolute deadline d_j
  - execution time c_j
- Platform constraint:
  - The energy overhead for turning on/off the system is L.
    - If the system is idle and turned on, the power consumption is assumed as a constant 1.
    - If the system is turned off, the power consumption is assumed 0.
  - The timing overhead can be considered as negligible. (This is not an issue for off-line scheduling)
- Output:
  - How to schedule the system to activate and de-activate (on and off) to minimize the energy consumption?

Simplest Case

- All the jobs are with unit execution time, i.e., $c_j = 1$ for all $J_j \in \mathcal{J}$.
- Energy overhead L is 1.

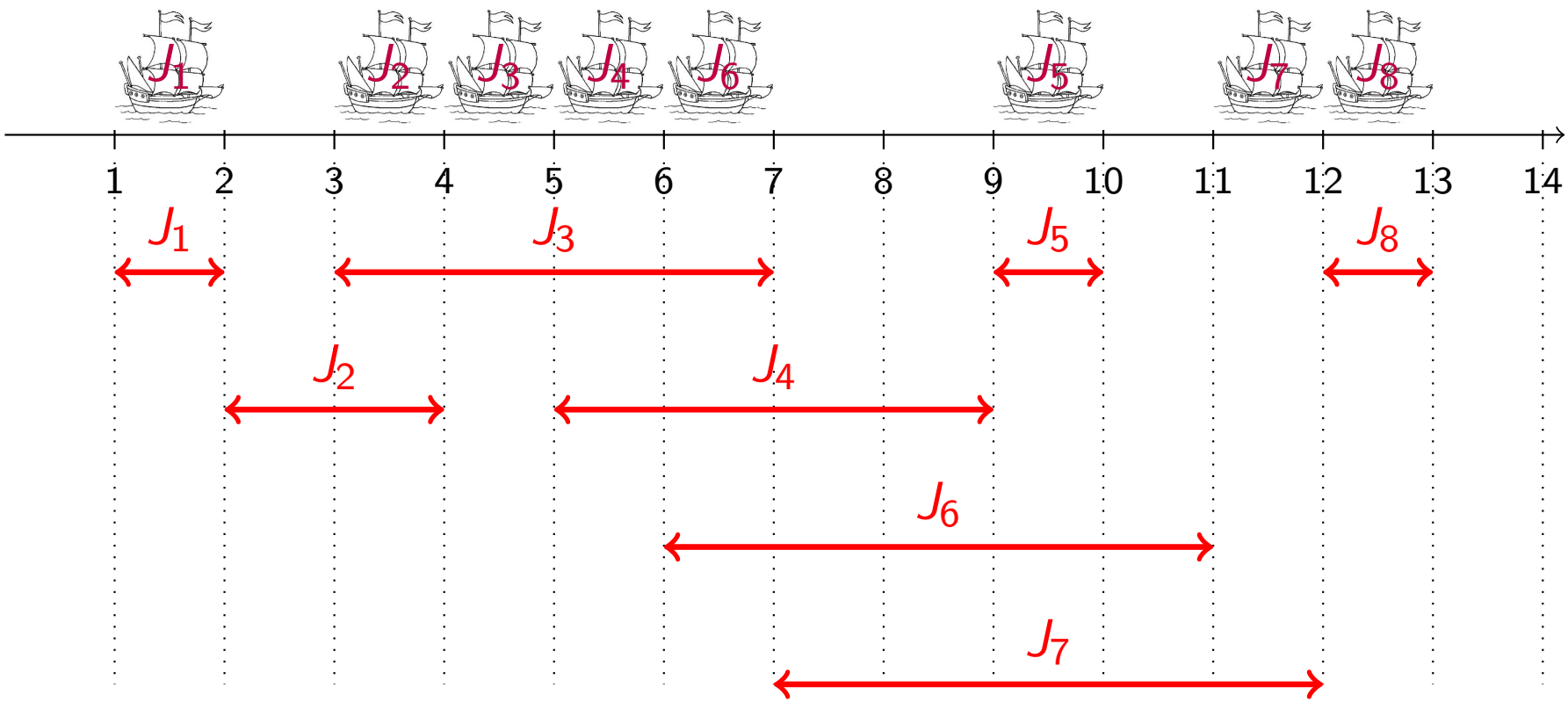
Posed as open by Sandy Irani, Kirk Pruhs: Algorithmic problems in power management. SIGACT News 36(2): 63-76 (2005)

A Traditional Problem for Chaining

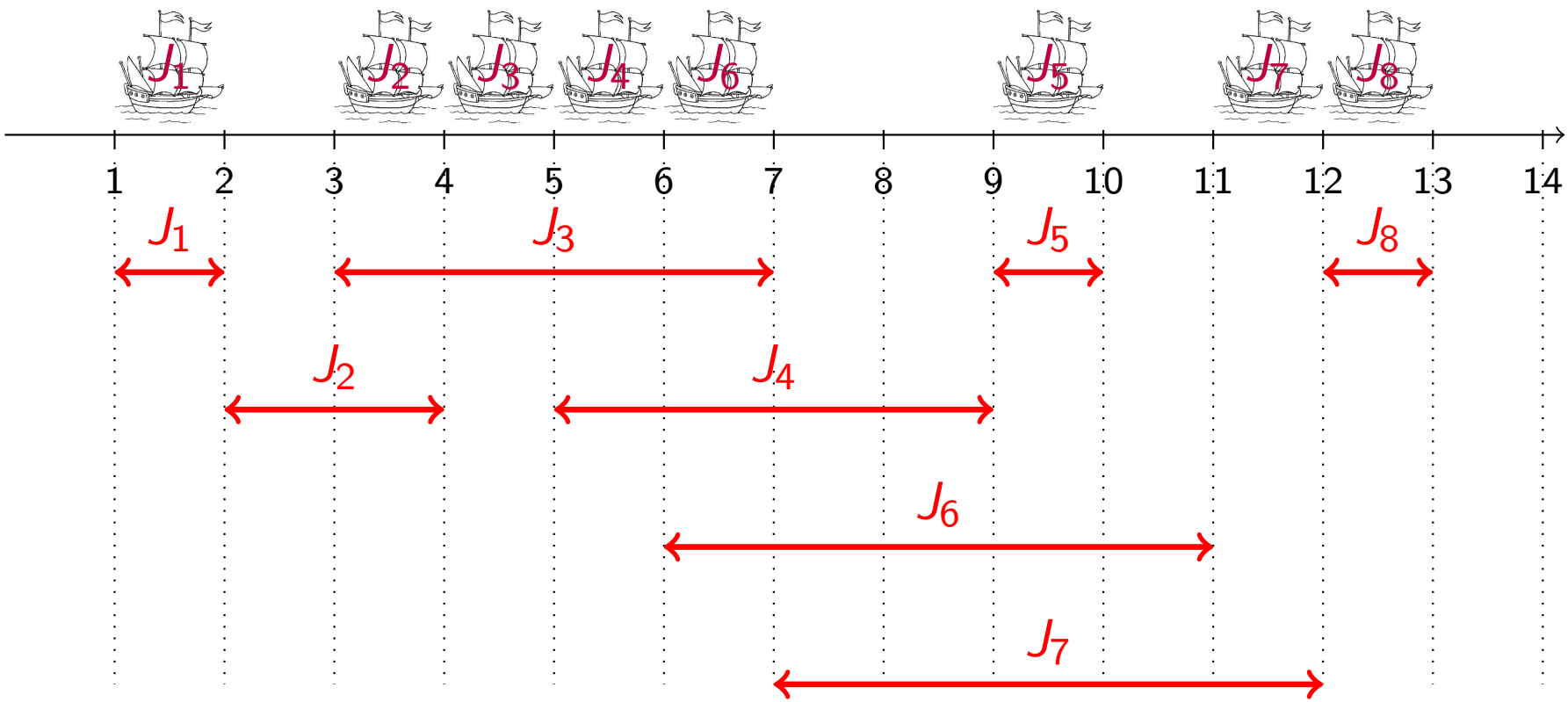
Chain the boats together for crossing Yangtze River, a famous story in China history in A.D. 208.

- Cao Cao was a general in northern China, who wanted to unify the country after he got the political power
- Sun Quan and Liu Bei were preparing in the other side of Yangtze River to defense
  - Unfortunately, Cao Cao's army couldn't get accustomed to the water, and many people became sick
  - Even though Cao Cao's army was stronger, the capability of soldiers would become very bad for fighting after crossing the river.
  - Cao Cao recruited some officers from southern China to train his army, but the progress was very limited.
  - Pang Tong suggested to Cao Cao to chain the boats together, so that soldiers are much more comfortable aboard.
    - If the boats have their range of movement in one dimension, how many chains (suppose they have the same cost) do we need?

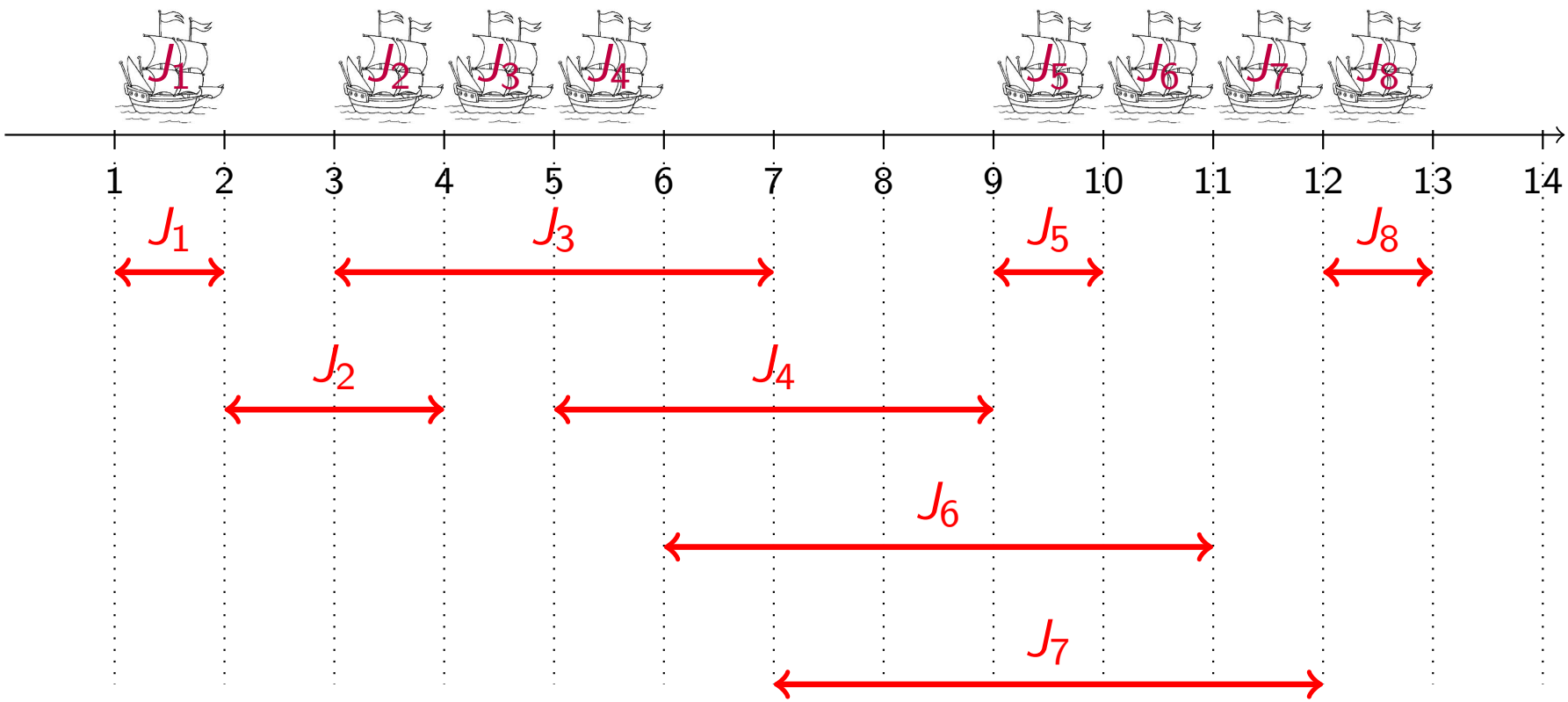
Chain the Boats



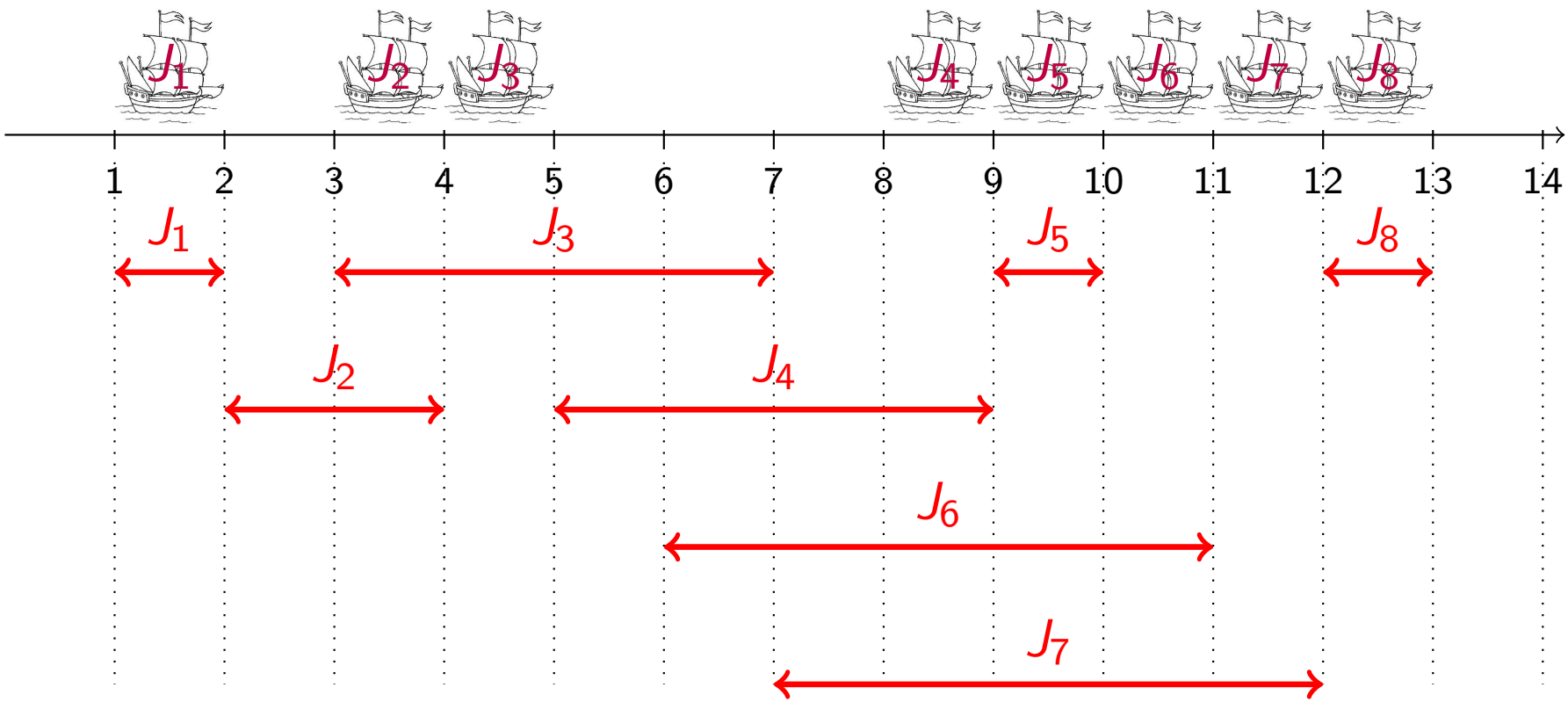
Chain the Boats



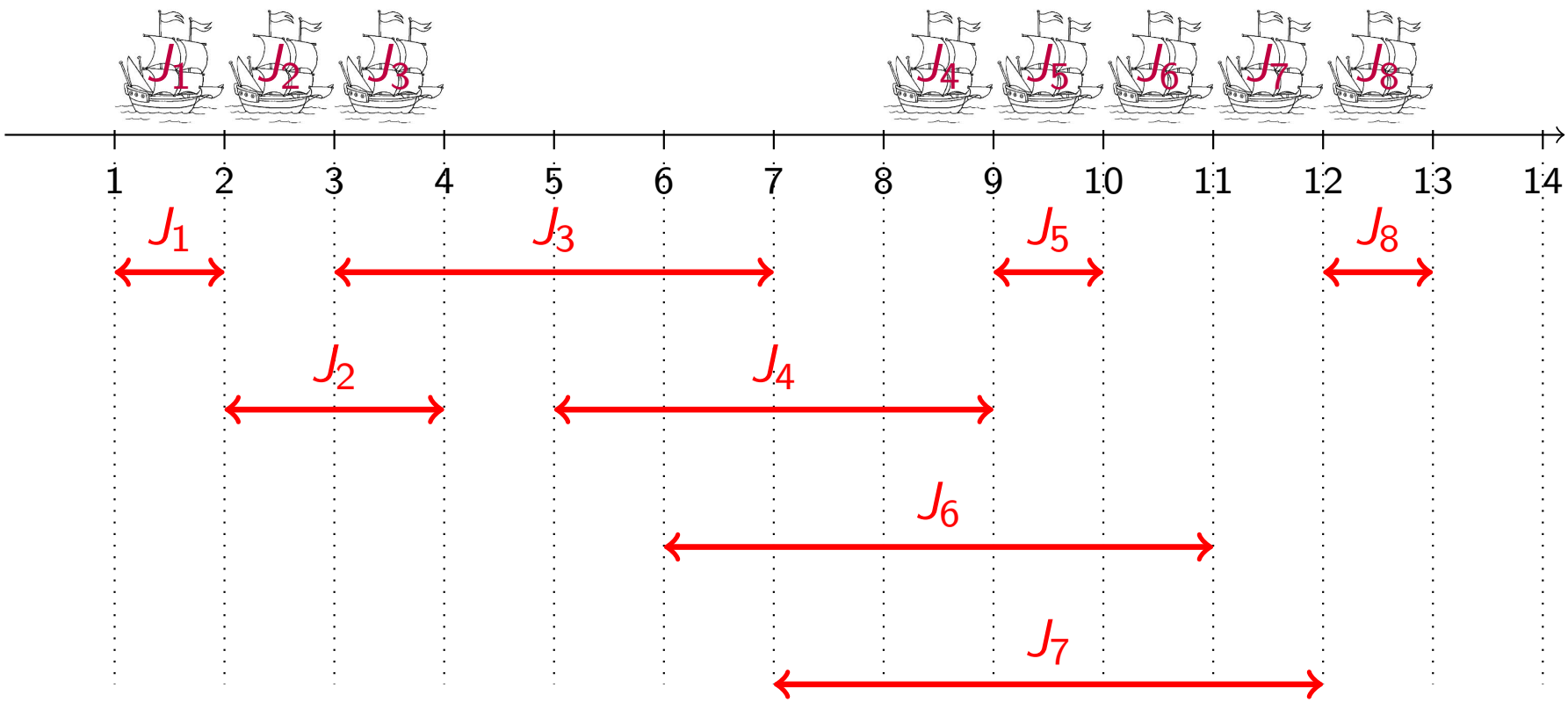
Chain the Boats



Chain the Boats



Chain the Boats

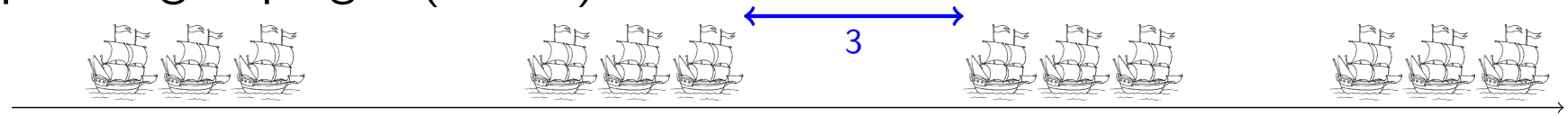


Only two chains are needed!

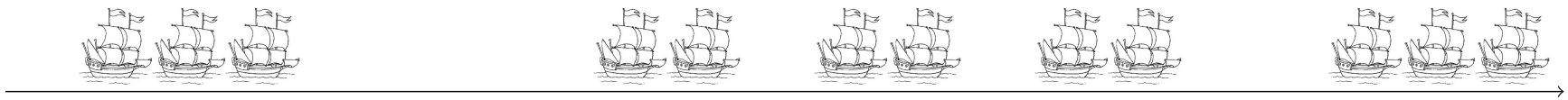
Chain the Boats

Intuition – why naive algorithms can't work ...

partial grouping 1 (better)



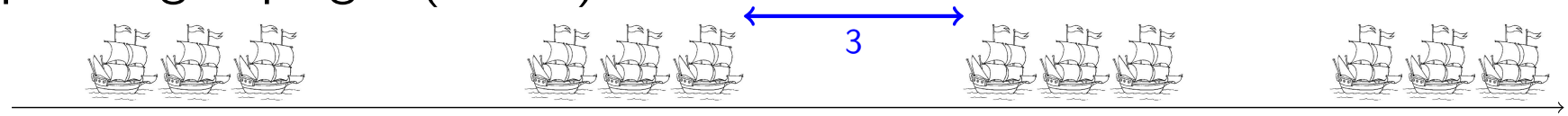
partial grouping 2



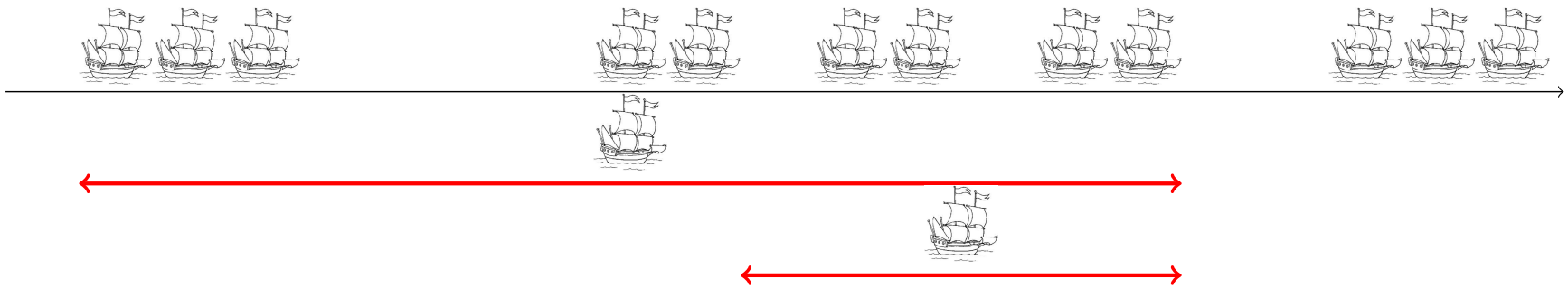
Chain the Boats

Intuition – why naive algorithms can't work ...

partial grouping 1 (better)



partial grouping 2

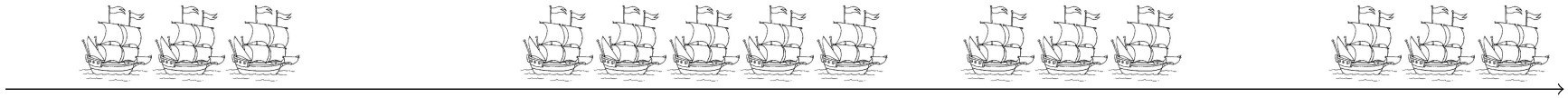


Now two more boats join....

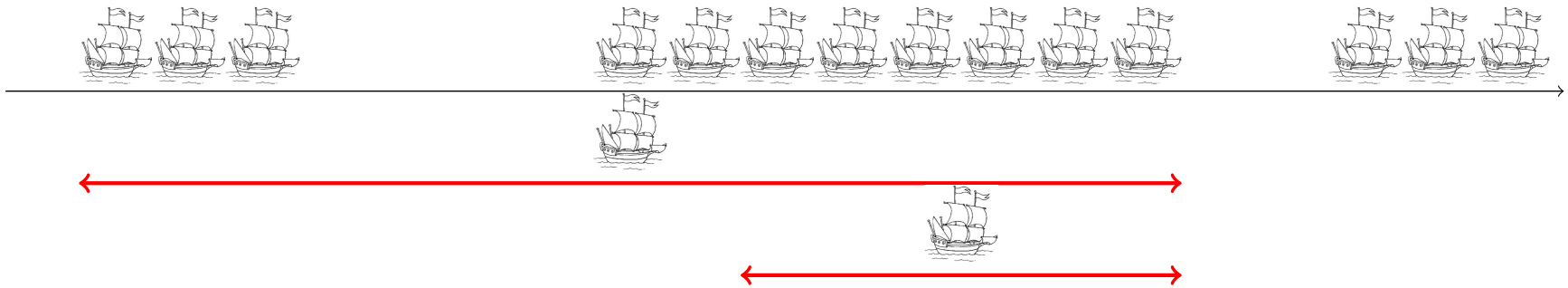
Chain the Boats

Intuition – why naive algorithms can't work ...

partial grouping 1



partial grouping 2 (becomes better)



Recap on Dynamic Programming

- Dynamic programming (roughly speaking)
  - The optimality of a problem can be achieved based on the optimal solutions of sub-programs.
  - The term actually comes from Control Theory, not computer science. Programming refers to the use of tables (arrays) to construct a solution.
  - Therefore, we usually solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table.
- One simple question: How to calculate the n-th Fibonacci number:
  - $F(n) = F(n-1) + F(n-2)$
  - $F(0) = 0$
  - $F(1) = 1$

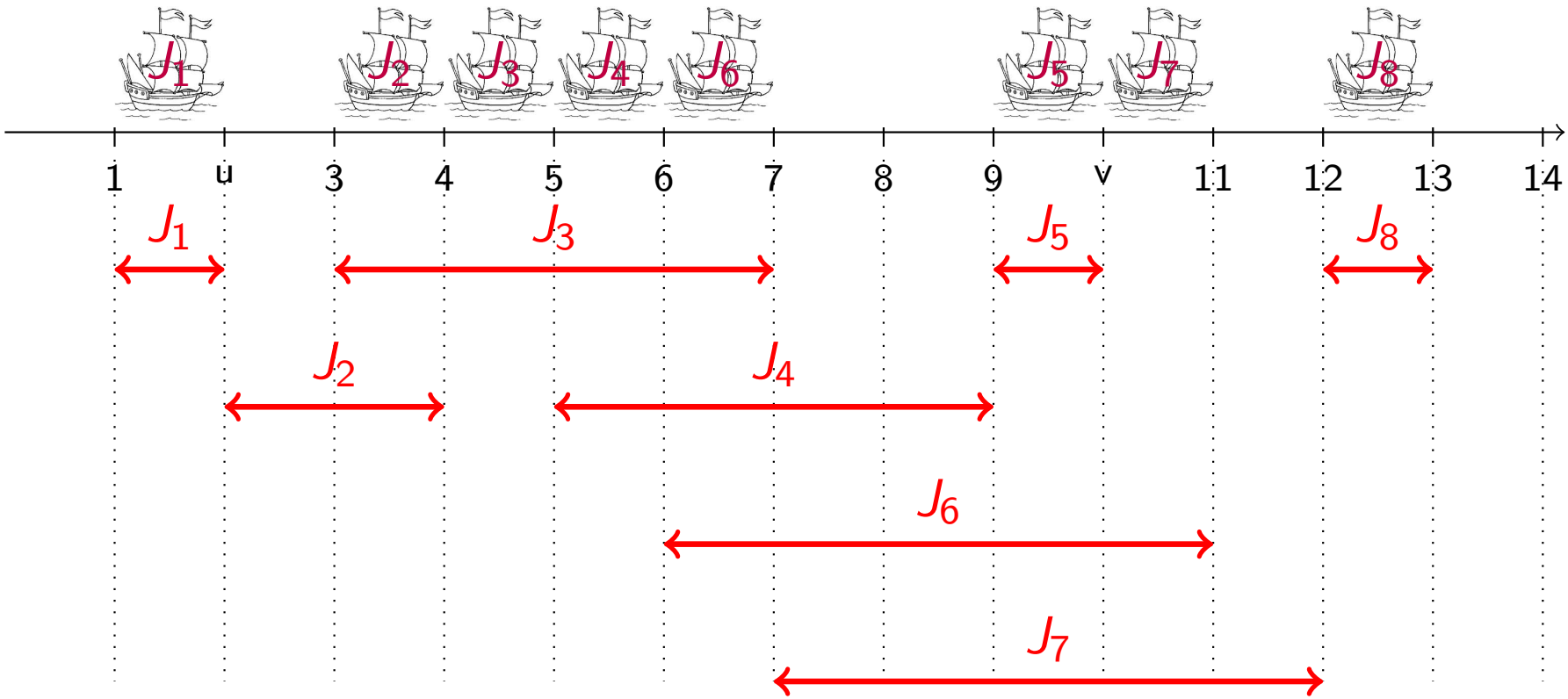
Recap on Dynamic Programming

- Dynamic programming (roughly speaking)
  - The optimality of a problem can be achieved based on the optimal solutions of sub-programs.
  - The term actually comes from Control Theory, not computer science. Programming refers to the use of tables (arrays) to construct a solution.
  - Therefore, we usually solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table.
- One simple question: How to calculate the n-th Fibonacci number:
  - $F(n) = F(n-1) + F(n-2)$
  - $F(0) = 0$
  - $F(1) = 1$
- Using dynamic programming: the complexity is $O(n)$
- Using recursive calls: the complexity is $O(2^n)$

Definition of Terms

$Boat_k(u, v) =$ all boats $J_j \in \{J_1, J_2, \ldots, J_k\}$ with $a_j \in [u, v)$

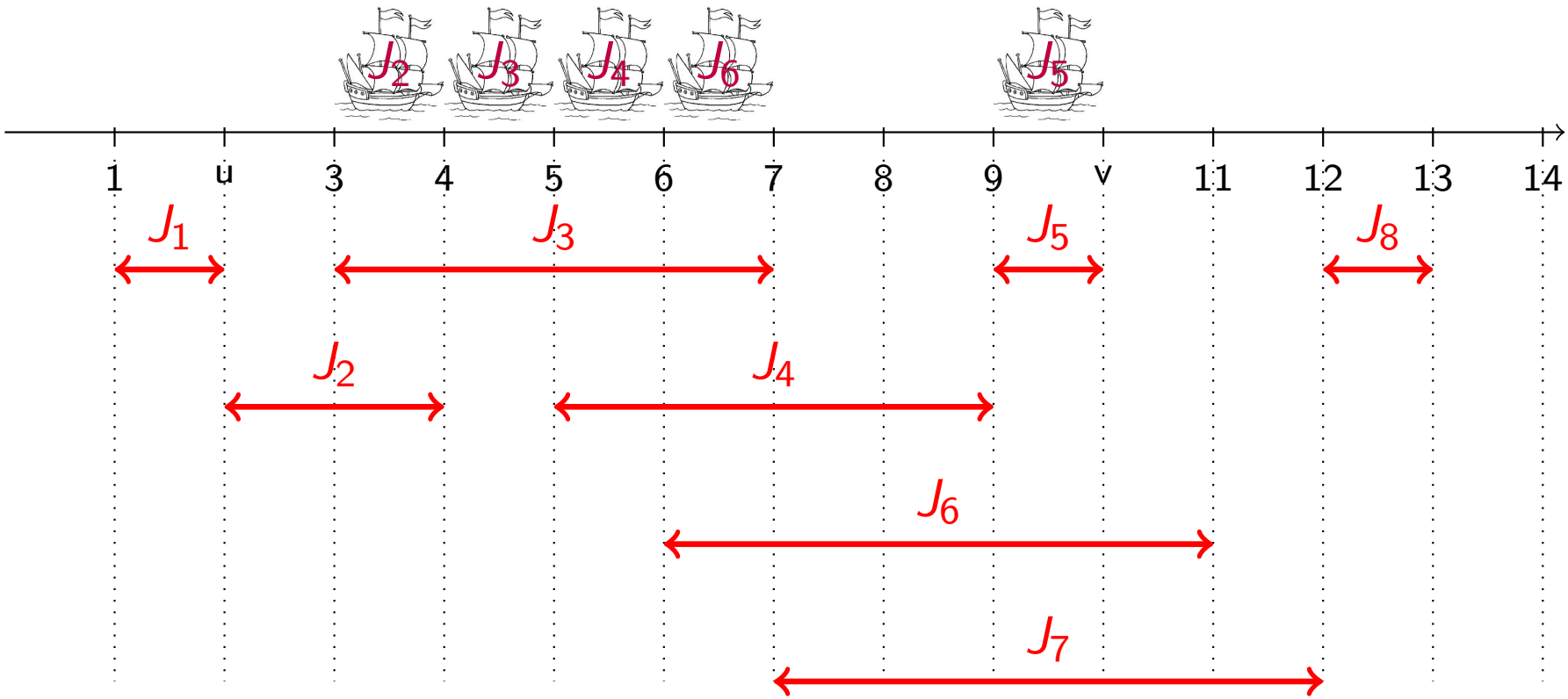
$Gaps_k(u, v) =$ min. number of gaps of $Boat_k(u, v)$ w.r.t $[u, v)$



Definition of Terms

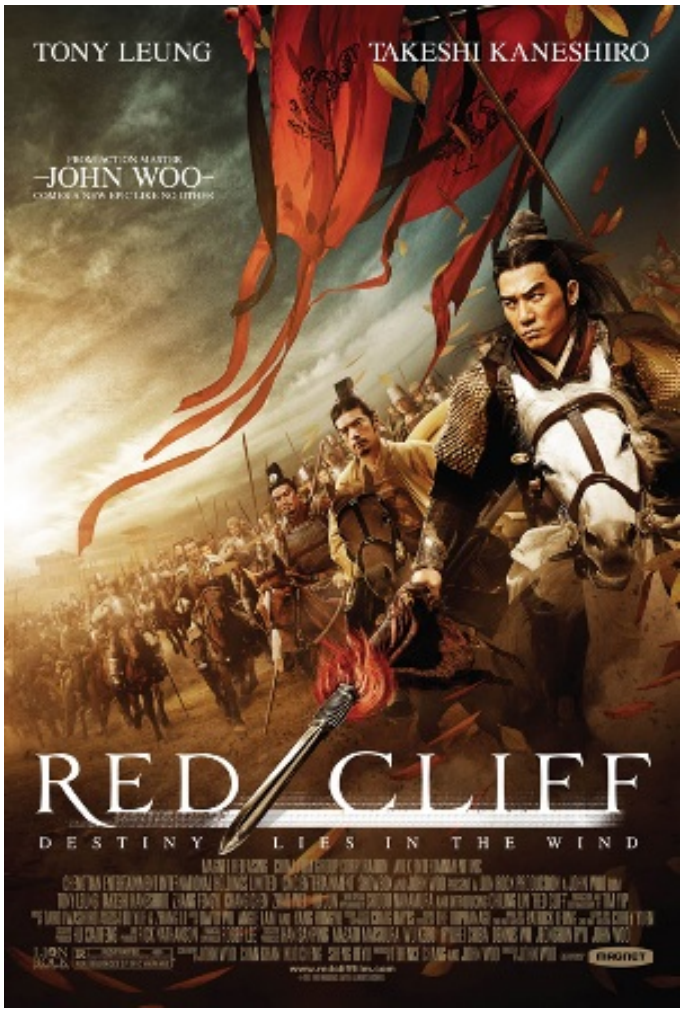
$Boat_k(u, v) =$ all boats $J_j \in \{J_1, J_2, \ldots, J_k\}$ with $a_j \in [u, v)$

$Gaps_k(u, v) =$ min. number of gaps of $Boat_k(u, v)$ w.r.t $[u, v)$



$Boat_6(2, 10) = \{J_2, J_3, J_4, J_5, J_6\}$, $Gaps_6(2, 10) = 2$,

Fire on the Boats



Poster from

http://en.wikipedia.org/wiki/File:Redcliffposter.jpg

- Cao Cao took Pang Tong's suggestion to chain the boats.
- The soldiers were much less sick, and the army thought they can easily conquer the other side of Yangtze River.
- Sun Quan and Liu Bei ordered Huang Gai to put fire on the chained boats.
- Cao Cao's army was defeated because of the chained boats sank....

Further Improvements

- The complexity can be reduced to $O(n^4)$ as well.
- The algorithm can be further improved to deal with general cases
  - Arbitrary execution time
  - $L \neq 1$

These are all presented in "Philippe Baptiste, Marek Chrobak, Christoph Dürr: Polynomial Time Algorithms for Minimum Energy Scheduling. ESA 2007: 136-150"

Problem Definition

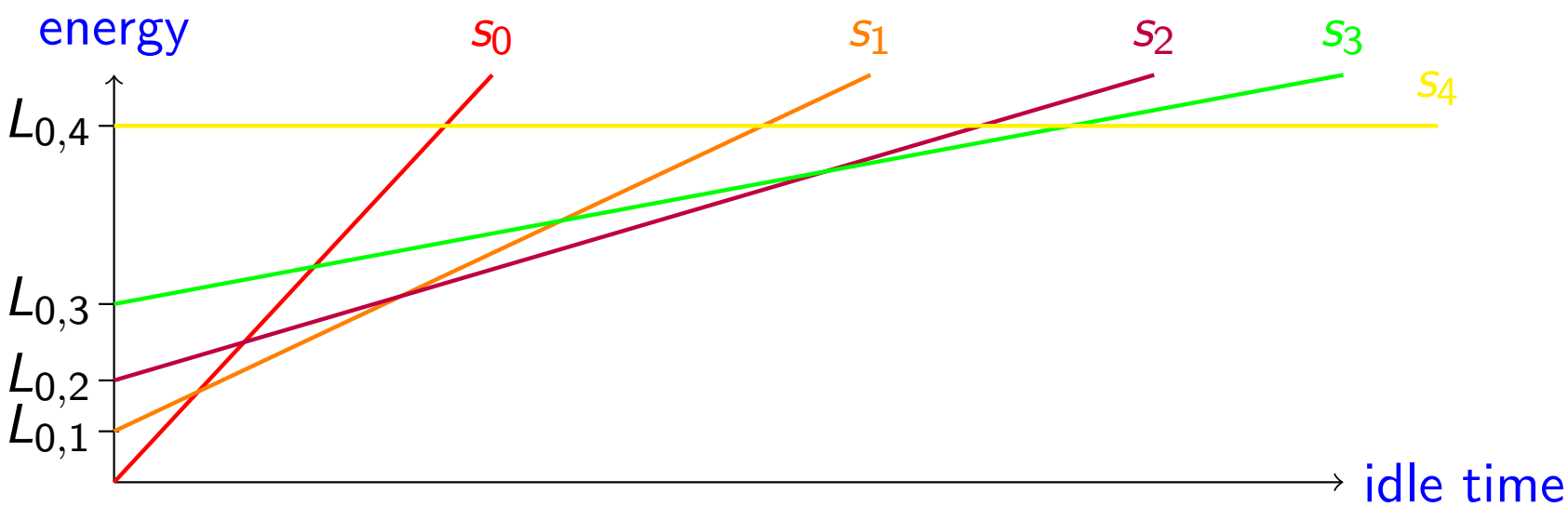
Problem Definition:

- When a system has nothing to work on at time t, how do we decide whether we keep the system active or turn off the system to reduce the energy consumption.
  - Karlin et al. study other problems, but is related to this topic when the system has only two states.
  - Augustine et al. consider general systems with multiple sleep states.
    - Multiple sleep states with general energy relations.

Multiple Power States

The platform (system) has a set of states $S = (s_0, s_1, \ldots, s_K)$

- The system is on (active) and idle at state s_0:
  - Power consumption P_0.
- The system is in lower power modes for idling:
  - Power consumption P_i at state s_i
- Power consumption is decreasing $P_0 > P_1 > \cdots > P_K$
- Energy consumption $L_{i,j}$ at idling power state s_i to power state s_j.



Outline

Traditional Problem for Skiing Beginners

Mr. Algorithmus visits Zermatt for his vocation and wants to enjoy skiing. He is not sure about how many days he will ski from now on.

- Rent the equipment is 60 CHF / day.
- Buy the equipment is 480 CHF.

Suppose that there is no cost for maintaining the equipment. What's the best strategy?

Traditional Problem for Skiing Beginners

Mr. Algorithmus visits Zermatt for his vocation and wants to enjoy skiing. He is not sure about how many days he will ski from now on.

- Rent the equipment is 60 CHF / day.
- Buy the equipment is 480 CHF.

Suppose that there is no cost for maintaining the equipment. What's the best strategy?

- Buy the equipment immediately on the first day
- Rent the equipment everyday
- Rent the equipment for the first few days and then buy the equipment if he wants to continue

Competitive Analysis for On-Line Algorithms

- Deterministic version:
  - Let A be a deterministic on-line algorithm. Algorithm A is said to have a ρ-competitive factor if for any input sequence σ

$$E(A(\sigma)) \leq \rho E(A^*(\sigma)) + a,$$

    where A^* is an optimal algorithm (or any offline algorithm) which knows the input sequence σ in advance, $E(A(\sigma))$ and $E(A^*(\sigma))$ are the objective functions of the results of A and A^* on σ, respectively, and a is a constant.
- Randomized version:
  - Let A be a randomized on-line algorithm. Algorithm A is said to have a ρ-competitive factor if for any input sequence σ

$$\overline{E}(A(\sigma)) \leq \rho E(A^*(\sigma)) + a,$$

    where A^* is an optimal algorithm (or any offline algorithm) any which knows the input sequence σ in advance, $\overline{E}(A(\sigma))$ is the expected objective function based on the randomized choices made by A, and a is a constant.

Skiing Rental Problem

Mr. Algorithmus visits Zermatt for his vocation and wants to enjoy skiing. He is not sure about how many days he will ski from now on.

- Rent the equipment is 60 CHF / day.
- Buy the equipment is 480 CHF.

A deterministic strategy:

- Rent the equipment for the first 8 days and then buy the equipment if he wants to continue
  - The break-even time is $480/60 = 8$ days.

Why is this strategy good?

Skiing Rental Problem

Mr. Algorithmus visits Zermatt for his vocation and wants to enjoy skiing. He is not sure about how many days he will ski from now on.

- Rent the equipment is 60 CHF / day.
- Buy the equipment is 480 CHF.

A deterministic strategy:

- Rent the equipment for the first 8 days and then buy the equipment if he wants to continue
  - The break-even time is $480/60 = 8$ days.

Why is this strategy good?

A 2-competitive on-line algorithm for skiing rental.

Proof for the 2-Competitive Factor

- If the request for not skiing comes before the break-even time, the rental cost of the on-line algorithm is the same as the optimal off-line decision.
- If the request for not skiing comes after the break-even time,
  - the rental cost of the on-line algorithm is 960.
  - the rental cost of the off-line optimal algorithm is to buy the equipment immediately with cost 480.

Proof for the 2-Competitive Factor

- If the request for not skiing comes before the break-even time, the rental cost of the on-line algorithm is the same as the optimal off-line decision.
- If the request for not skiing comes after the break-even time,
  - the rental cost of the on-line algorithm is 960.
  - the rental cost of the off-line optimal algorithm is to buy the equipment immediately with cost 480.

When a system has nothing to work at time t and the next job comes at time t' (this value is unknown until t' reaches) with $t' > t$,

- if $t' \leq t + \frac{L_{0,1}}{P_0 - P_1}$, we keep the system active from $[t, t')$;
  - This is the same as the offline optimal solution.
- otherwise, we keep the system active from $[t, t + \frac{L_{0,1}}{P_0 - P_1})$ and turn the system to state s_1 at time $t + \frac{L_{0,1}}{P_0 - P_1}$.
  - This is at most twice of the objective function of the offline optimal solution.

Randomization

- Randomization may help.
- Recall that the problem is to decide how to sleep when the system becomes idle at time t.
  - 2-competitive by waiting for the break-even time and the sleep mode.
  - $\frac{e}{e-1}$-competitive factor by choosing a proper distribution to decide when to enter the sleep mode at time t.
- For the simplification of presentations, we set
  - P_1 to 0 and P_0 to $P_0 - P_1$.

Distribution

Let C be the break-even time $\frac{L_{0,1}}{P_0}$, and $\pi(x)$ be the density function of the time before the system enters the sleep mode.

- $\pi(x) = 0$ if $x > C$
- Suppose that the next event comes at time $t + \sigma$.
- The expected energy consumption is

$$E(\pi_\sigma) = \int_0^\sigma (P_0 x + L_{0,1})\pi(x)dx + \int_\sigma^\infty P_0 \sigma \pi(x)dx.$$

Distribution

Let C be the break-even time $\frac{L_{0,1}}{P_0}$, and $\pi(x)$ be the density function of the time before the system enters the sleep mode.

- $\pi(x) = 0$ if $x > C$
- Suppose that the next event comes at time $t + \sigma$.
- The expected energy consumption is

$$E(\pi_\sigma) = \int_0^\sigma (P_0 x + L_{0,1})\pi(x)dx + \int_\sigma^\infty P_0 \sigma \pi(x)dx.$$

Expected energy consumption when the system decides to sleep after idling x amount of time

Distribution

Let C be the break-even time $\frac{L_{0,1}}{P_0}$, and $\pi(x)$ be the density function of the time before the system enters the sleep mode.

- $\pi(x) = 0$ if $x > C$
- Suppose that the next event comes at time $t + \sigma$.
- The expected energy consumption is

$$E(\pi_\sigma) = \int_0^\sigma (P_0 x + L_{0,1})\pi(x)dx + \int_\sigma^\infty P_0 \sigma \pi(x)dx.$$

Expected energy consumption when the system decides to sleep after idling x amount of time

Expected energy consumption when the system decides to sleep after idling $x > \sigma$ amount of time

Distribution

Let C be the break-even time $\frac{L_{0,1}}{P_0}$, and $\pi(x)$ be the density function of the time before the system enters the sleep mode.

- $\pi(x) = 0$ if $x > C$
- Suppose that the next event comes at time $t + \sigma$.
- The expected energy consumption is

$$E(\pi_\sigma) = \int_0^\sigma (P_0 x + L_{0,1})\pi(x)dx + \int_\sigma^\infty P_0 \sigma \pi(x)dx.$$

Expected energy consumption when the system decides to sleep after idling x amount of time

Expected energy consumption when the system decides to sleep after idling $x > \sigma$ amount of time

What is the distribution for minimizing $E(\pi_\sigma)$?

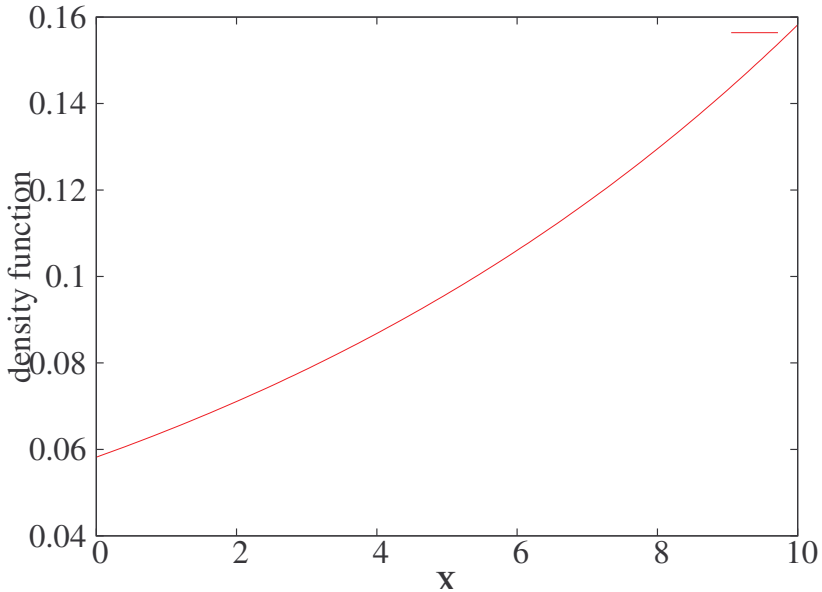
Competitive Factor (Analysis Omitted)

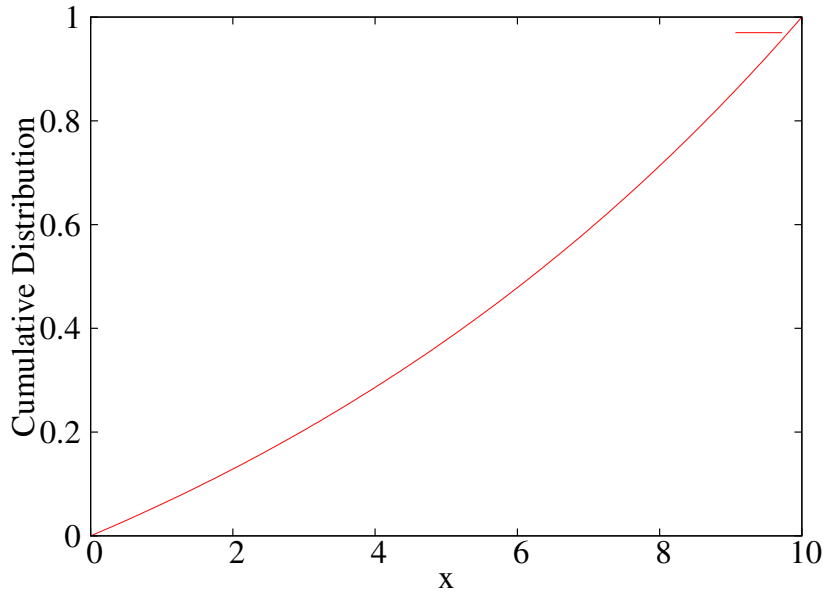
When the system is idle at time t, if the decision is made by using the density function $\pi(x)$ as follows to decide the idle time awaiting for sleeping after time t,

$$\begin{cases} \pi(x) = \frac{1}{C(e-1)} e^{\frac{x}{C}} & x \leq C, \\ \pi(x) = 0 & x > C, \end{cases}$$

the competitive factor is $\frac{e}{e-1}$, since α is $\frac{1}{e-1}$.

Illustration when $C = 10$





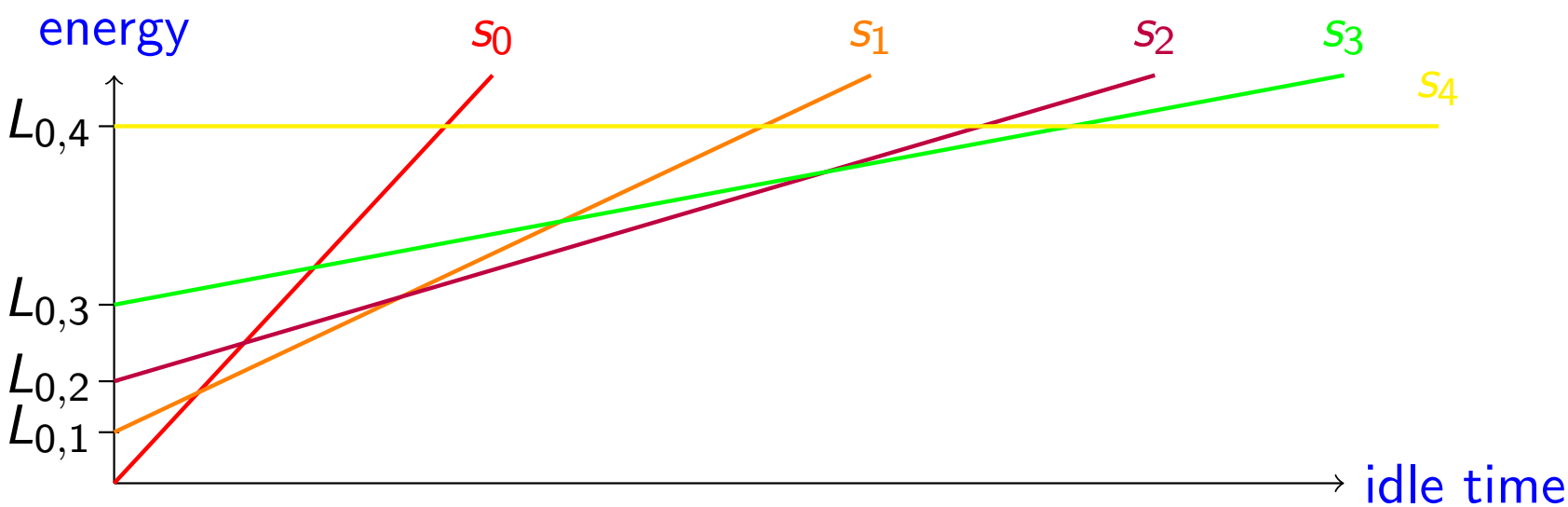
Outline

Additive Transition Systems

A system is called additive if the power states satisfy the following property:

- $L_{i,\ell} + L_{\ell,j} = L_{i,j}, \forall 0 \leq i < \ell < j \leq K$.

Therefore, $L_{0,i} < L_{0,i+1}$. The function $H(x)$ for optimal state transition, when the idle time is x, is a piece-wise linear function, and can be derived in polynomial time of K.

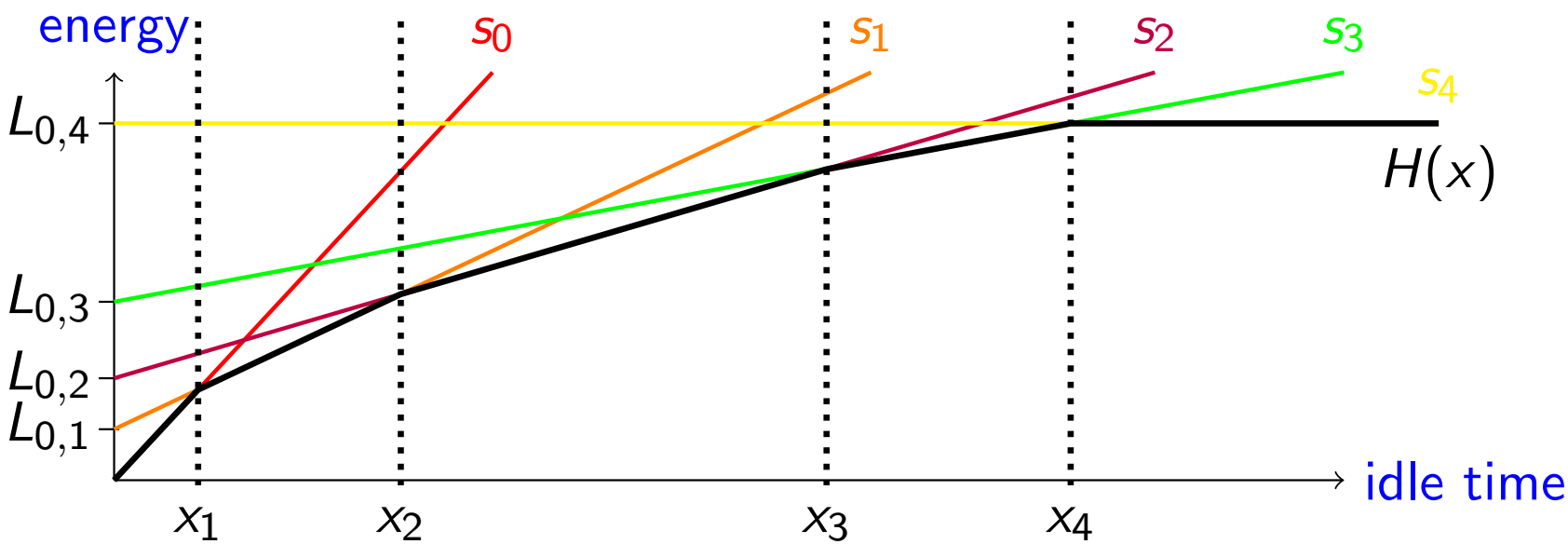


Additive Transition Systems

A system is called additive if the power states satisfy the following property:

- $L_{i,\ell} + L_{\ell,j} = L_{i,j}, \forall 0 \leq i < \ell < j \leq K$.

Therefore, $L_{0,i} < L_{0,i+1}$. The function $H(x)$ for optimal state transition, when the idle time is x, is a piece-wise linear function, and can be derived in polynomial time of K.



Function $H(x)$

If the system has idle time x, an optimal optimal solution will transit from state s_0 to a certain state s_i such that $L_{0,i} + xP_i$ is minimized. Therefore,

Function $H(x)$

If the system has idle time x, an optimal optimal solution will transit from state s_0 to a certain state s_i such that $L_{0,i} + xP_i$ is minimized. Therefore,

$$H(x) = \min_i \{L_{0,i} + xP_i\}.$$

Suppose that the optimal offline solution enters s_i for any $x \in (x_i, x_{i+1}]$, then

- $x_0 = 0, x_{K+1} = \infty$, and
- $x_i = \frac{L_{0,i} - L_{0,i-1}}{P_{i-1} - P_i}$.

Competitive Factors

- Suppose that $H(x)$ is given and the state transition is from $s_0 \to s_1 \to s_2 \to \cdots \to s_K$ when the system has idled for time interval length $x_1, x_2, x_3, \ldots, x_K$.
  - The deterministic algorithm by following the state transition function $H(x)$ from s_0 is a 2-competitive algorithm
  - The randomized algorithm has a $\frac{e}{e-1}$-competitive factor
    - When the system idles at time t at state s_i for decision making of state transition, we use the following density function to decide when the system should enter the state s_{i+1}, where C_i is $\frac{L_{i,i+1}}{P_i - P_{i+1}}$:

$$\begin{cases} \pi(x) = \frac{1}{C_i(e-1)} e^{\frac{x}{C_i}} & x \le C_i \\ \pi(x) = 0 & x > C_i \end{cases}$$

    - The system decides how to change the state at time $t + C_i$ if no event comes between t and $t + C_i$ (i.e., t is now $t + C_i$ and the idle state becomes s_{i+1}).

Competitive Factors

- Suppose that $H(x)$ is given and the state transition is from $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_K$ when the system has idled for time interval length $x_1, x_2, x_3, \ldots, x_K$.
  - The deterministic algorithm by following the state transition function $H(x)$ from s_0 is a 2-competitive algorithm
  - The randomized algorithm has a $\frac{e}{e-1}$-competitive factor
    - When the system idles at time t at state s_i for decision making of state transition, we use the following density function to decide when the system should enter the state s_{i+1}, where C_i is $\frac{L_{i,i+1}}{P_i - P_{i+1}}$:

$$\begin{cases} \pi(x) = \frac{1}{C_i(e-1)} e^{\frac{x}{C_i}} & x \leq C_i \\ \pi(x) = 0 & x > C_i \end{cases}$$

    - The system decides how to change the state at time $t + C_i$ if no event comes between t and $t + C_i$ (i.e., t is now $t + C_i$ and the idle state becomes s_{i+1}).

Sandy Irani, Rajesh K. Gupta, Sandeep K. Shukla: Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Savings States. DATE 2002: 117-123

Non-Additive Power States

- Power consumption is decreasing $P_0 > P_1 > \cdots > P_k$.
- Energy consumption $L_{i,j}$ at idling power state s_i to power state s_j.

Non-Additive Power States

- Power consumption is decreasing $P_0 > P_1 > \cdots > P_k$.
- Energy consumption $L_{i,j}$ at idling power state s_i to power state s_j.

The deterministic algorithm by using only a subset of states and following the state transition function $H(x)$ from s_0 when time progresses is a $3 + 2\sqrt{2} \approx 5.828$-competitive algorithm.

John Augustine, Sandy Irani, Chaitanya Swamy: Optimal Power-Down Strategies. SIAM J. Comput. 37(5): 1499-1516 (2008) (conference: FOCS 2004: 530-539)