Flow Algorithms for Two Pipelined Filtering Problems

Anne Condon, University of British Columbia
Amol Deshpande, University of Maryland
Lisa Hellerstein, Polytechnic University, Brooklyn
Ning Wu, Polytechnic University, Brooklyn

Pipelined Filter Ordering

- Query processors commonly need to evaluate complex predicates against relations
  - Eg., on an *employee* relation:

    ((salary > 120000) AND (*status = 2*)) OR *(educationlevel = 3)*;

    (age < 30) AND hasPatents(emp) AND hasWrittenBooks(emp);
- Pipelined filter ordering problem

  Decide the order in which to apply the individual predicates (filters) to the tuples
- We will focus on evaluating *conjunctive predicates* (containing only ANDs)

Example Query

select *
from *employee*
where *age* < *30* and
    salary < *100000* and
    zipcode = *20001*

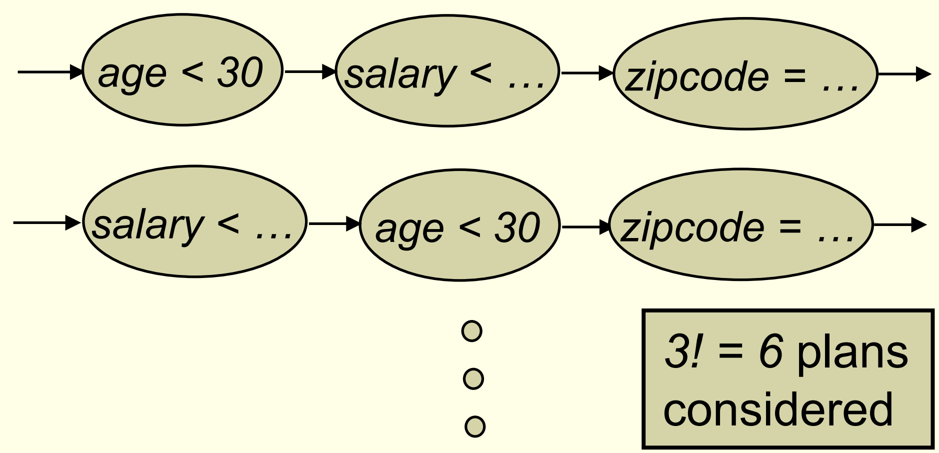
Classical Pipelined Filter Ordering

- Given:
  - A set of *independent* filters, $pred_i$
  - *cost* of applying each filter, c_i
  - *selectivity* of each filter, p_i
- Find:
  - the optimal *serial* execution order for applying the filters that minimizes the *total execution cost on a single processor*

Example Query

select *
from *employee*
where *age < 30* and
salary < 100000 and
zipcode = 20001

Serial Plans Considered



Classical Pipelined Filter Ordering

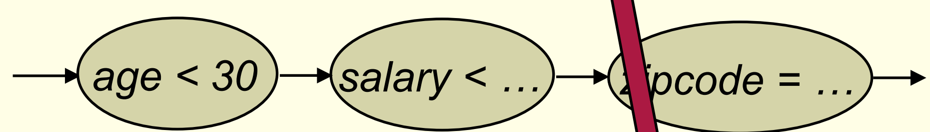
- Given:
  - A set of *independent* filters, $pred_i$
  - *cost* of applying each filter, c_i
  - *selectivity* of each filter, p_i
- Find:
  - the optimal *serial* execution order for applying the filters that minimizes the *total execution cost on a single processor*

Independence Assumption

Example Query

select *
from *employee*
where *age < 30* and
salary < 100000 and
zipcode = 20001

Execution cost of a plan



costs	c_1		c_2		c_3
selectivities	p_1		p_2		p_3
Expected cost per tuple	c_1	+	$p_1 c_2$	+	$p_1 p_2 c_3$

Classical Pipelined Filter Ordering

- Algorithm for *independent filters* [KBZ'86]
  - Apply the filters in the increasing order of:

    $$(1 - p_i) / c_i$$
  - $O(n \log (n))$

- Correlated filters ?
  - NP-hard under several formulations
    - E.g. when asked to find the best order for a given set of tuples
  - *4-Approx greedy* algorithm [BMMNW'04]

Pipelined Filter Ordering

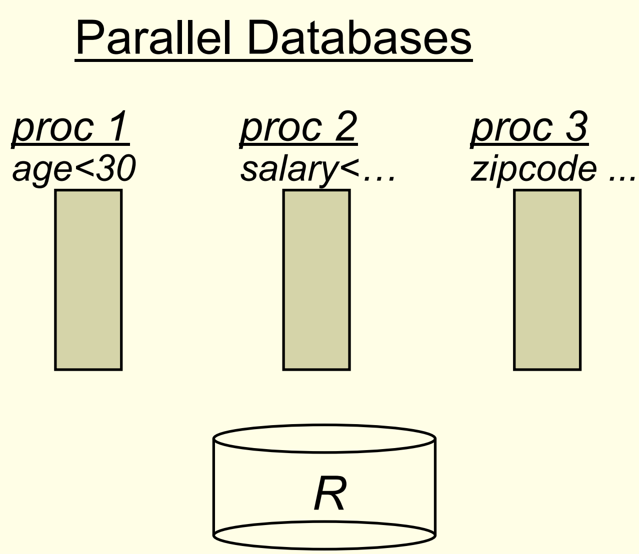
- Complex expensive predicates
  - E.g. *pointInPolygon(x, y, P), hasPatents(emp)??*
- Many join queries reduce to this problem
  - E.g. queries posed against a *star schema*
- Increasing interest in recent years
  - Data streams [AH'00, BMMNW'04]
  - Sensor Networks [DGMH'05]
  - Web indexes [CDY'95, EHJKMW'96, GW'00]
  - Web services [SM'06]
- Similar to many problems in other domains
  - Sequential testing (e.g. for fault detection) [SF'01, K'01]
  - Learning with attribute costs [KKM'05]

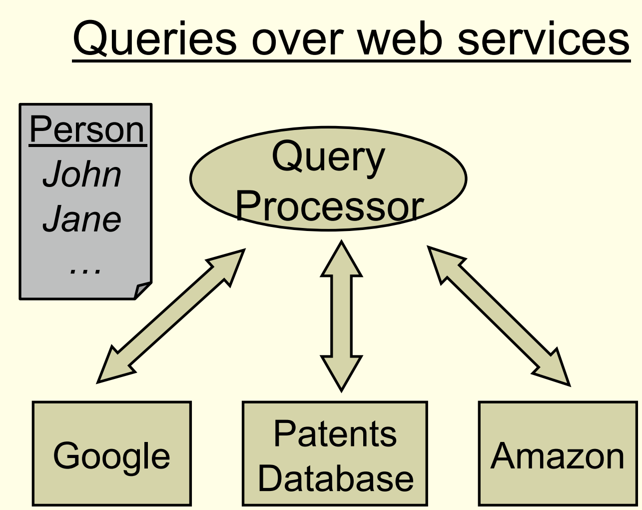
Outline

- Introduction
- Problem1: Max-throughput problem
  - Maximize *throughput* (tuples processed per unit time) in a parallel environment
- Problem 2: Adversarial type, Single Tuple
  - Minimize *multiplicative regret* in an adversarial setting

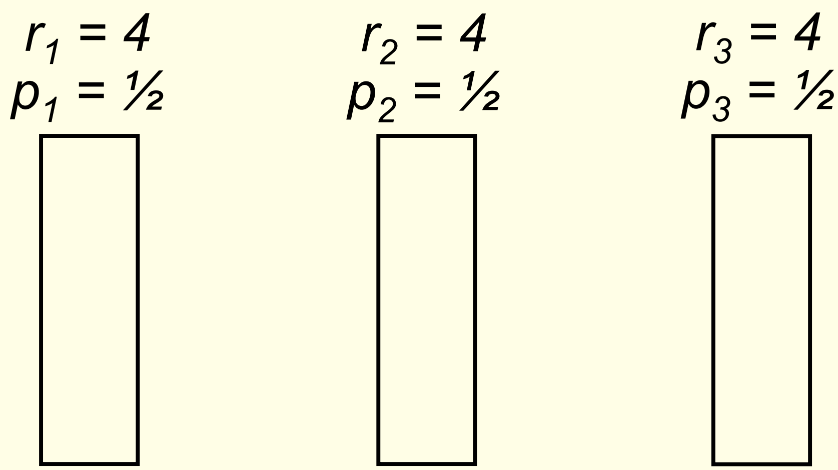
Problem 1: Max-throughput Problem

- *n independent* filters executed *in parallel* by *n operators,*
  $O_1, \ldots, O_n$
- Given:
  - *selectivities* of the filters, p_i
  - *rate limits* of the operators, r_i
- *Goal: Maximize the number of tuples processed per time unit*

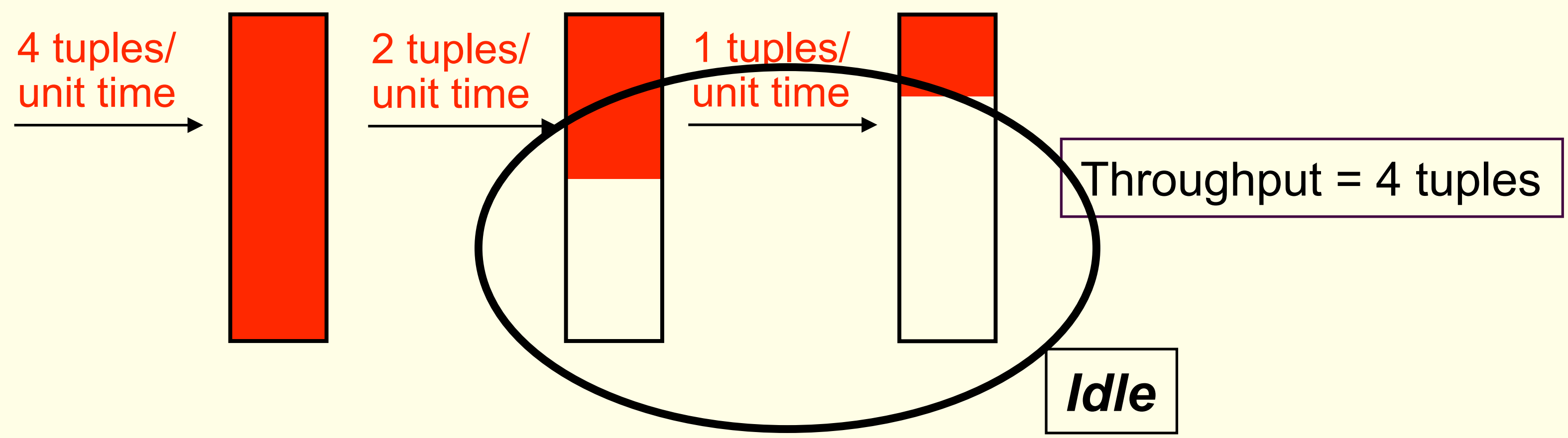




Max-throughput Problem

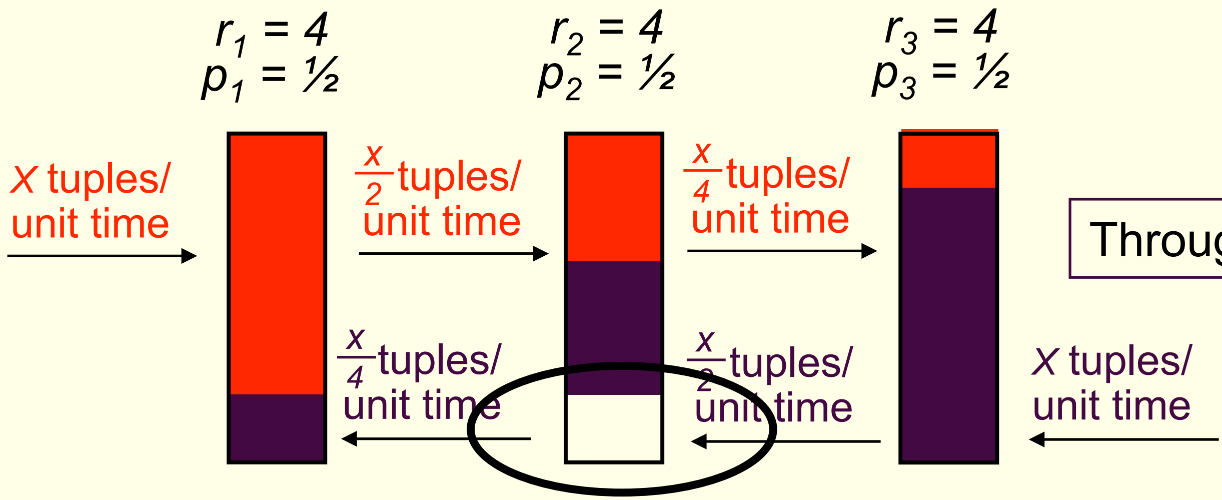


Best "single" execution order ?



Max-throughput Problem

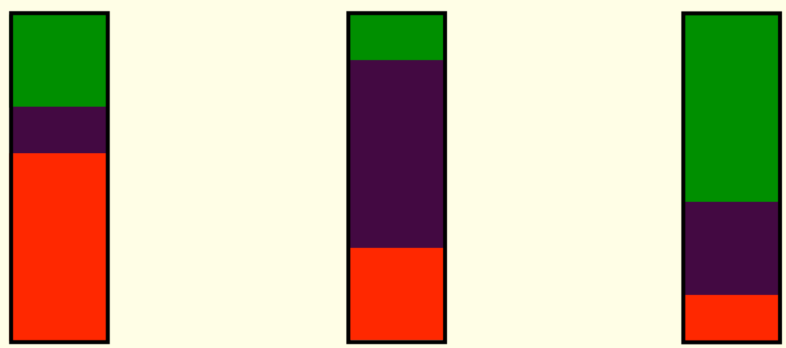
Use two orders: (1, 2, 3) and (3, 2, 1)



$X = \frac{16}{5}$

Throughput = $\frac{32}{5}$ tuples

Send Y tuples through (1, 2, 3), (2, 3, 1), and (3, 1, 2)



$Y = \frac{16}{7}$

Throughput = $\frac{48}{7}$ tuples

Best possible ??

Max-throughput Problem

- Definition:
  - Saturation = an operator is processing at its capacity
- Lemma: <u>Full saturation → optimality</u>
- Proof:

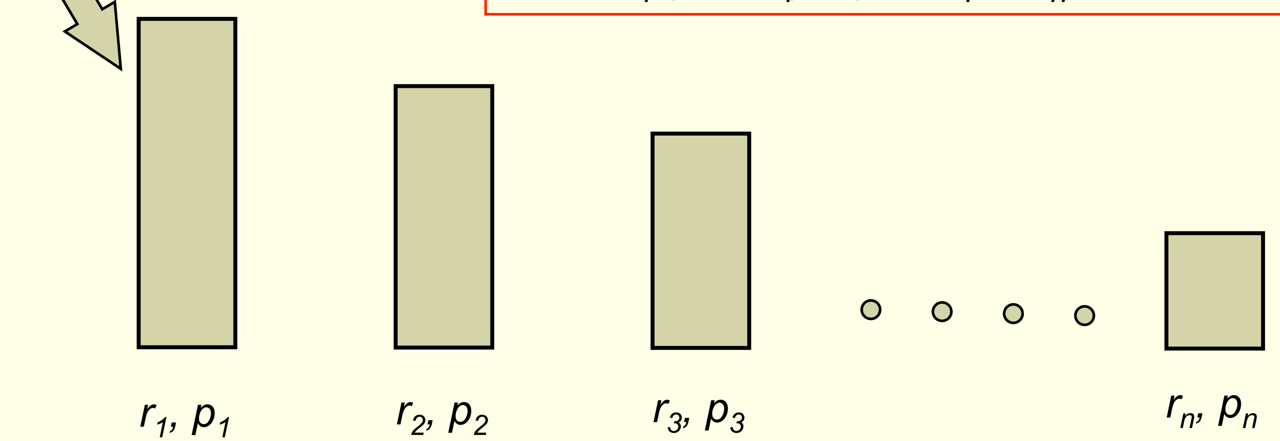
Total tuples rejected in unit time
$= \Sigma\, r_i\,(1 - p_i)$

Processes r_1 tuples
Rejects $r_1(1-p_1)$ tuples

If throughput equal to K,
then total tuples rejected in unit time
$= K\,(1 - p_1 \ldots p_n)$

Combined rejection probability =
$(1 - p_1 p_2 \ldots p_n)$

$\boxed{K = \Sigma\, r_i\,(1 - p_i) \,/\, (1 - p_1 \ldots p_n) = \text{Constant}}$



<u>Saturated steady state – Throughput *K*</u>

Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
- Adversarial type, Single Tuple

Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
    - Equal rates, unequal selectivities
    - Unequal rates and selectivities
      - Two operators
      - General case
  - Routing and queuing issues
- Adversarial type, Single Tuple

Summary of results

- Max-throughput problem
  - *O(n)* algorithm to find the maximal achievable throughput, if rates given in sorted order
  - $O(n^2)$ algorithm to find the optimal solution
  - Previously known best algorithms [Kodialam'01]
    - From sequential testing literature
    - $O(n^2)$ and $O(n^3 \log n)$ respectively

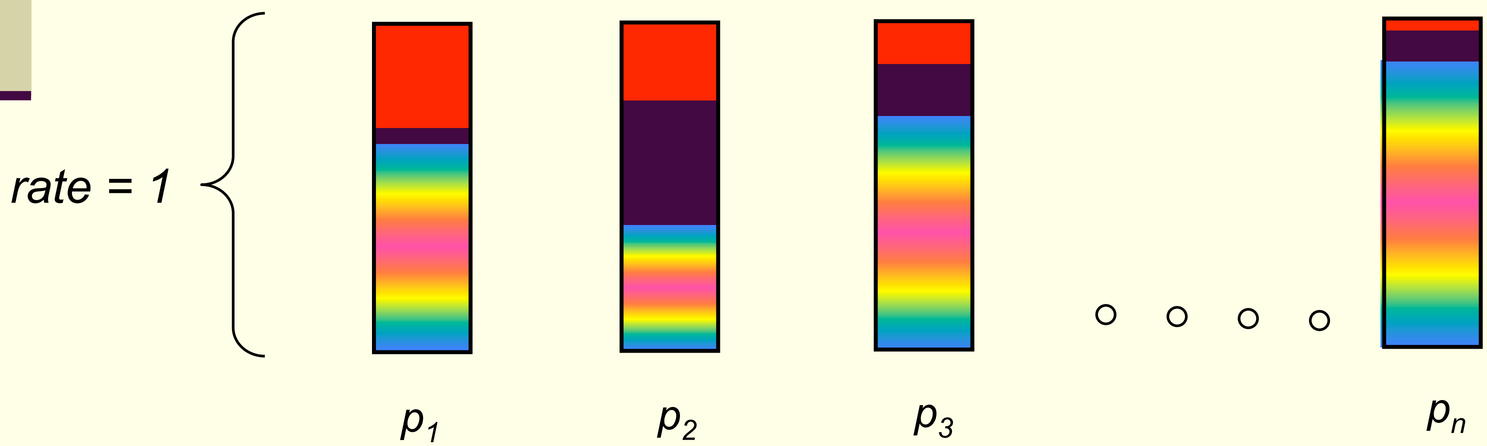
Equal rates, unequal selectivities

- Closed form *saturating* solution using *cyclic permutations*
- Send:

  $$(1 - p_{j-1}) / (n - \Sigma p_i)$$

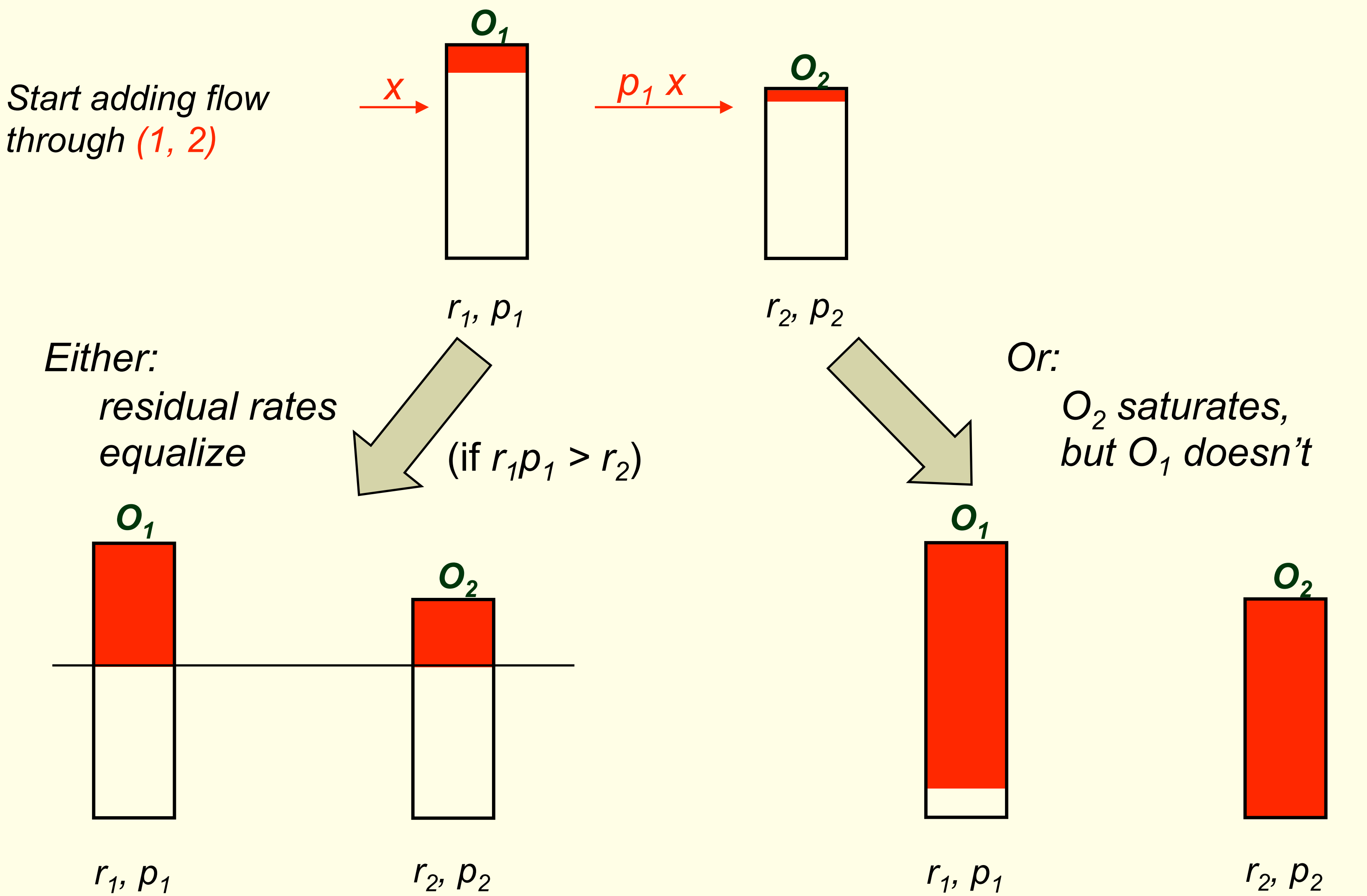
  tuples through permutation:

  $$(j, j+1, j+2, \ldots, n, 1, \ldots, j-1)$$



Unequal rates: Two Operators

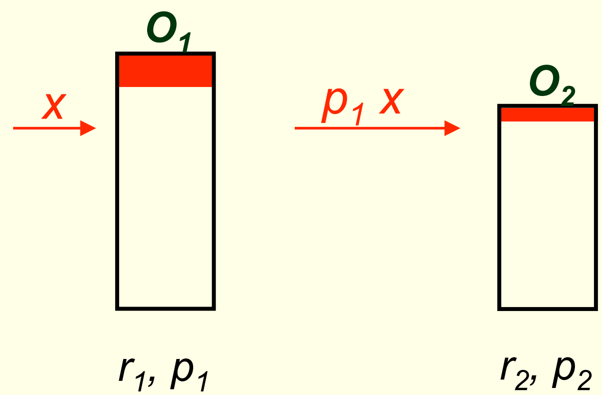
Two operators with unequal rates ($r_1 > r_2$)



Unequal rates: Two Operators

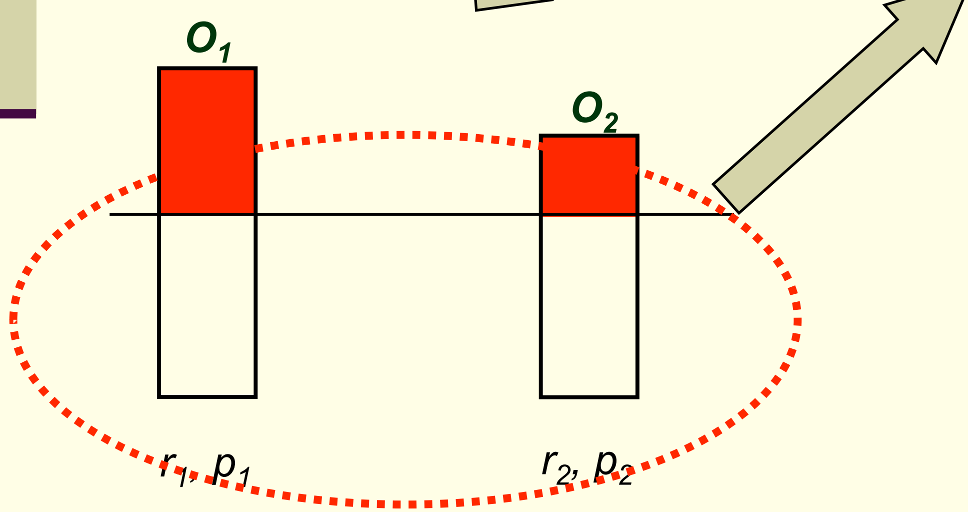
Two operators with unequal rates ($r_1 > r_2$)

Start adding flow through (1, 2)



Either:
residual rates equalize

(if $r_1 p_1 > r_2$)



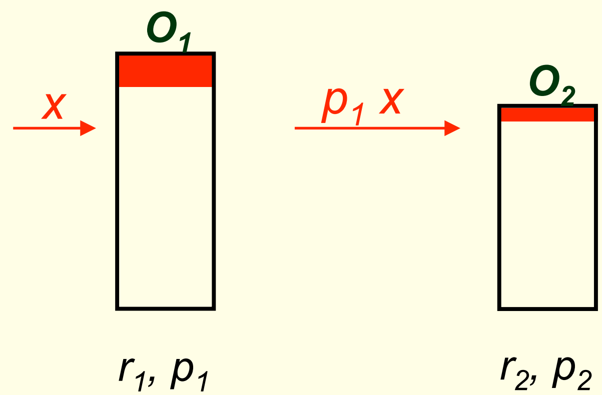
Use the equal-rates algorithm to solve the residual problem

<u>Two part solution:</u>
During every time unit,
1. Send x tuples through *(1, 2)*
2. Send y tuples total through cylic permutations: *(1, 2)* and *(2, 1)* (appropriately divided)

Unequal rates: Two Operators

Two operators with unequal rates ($r_1 > r_2$)

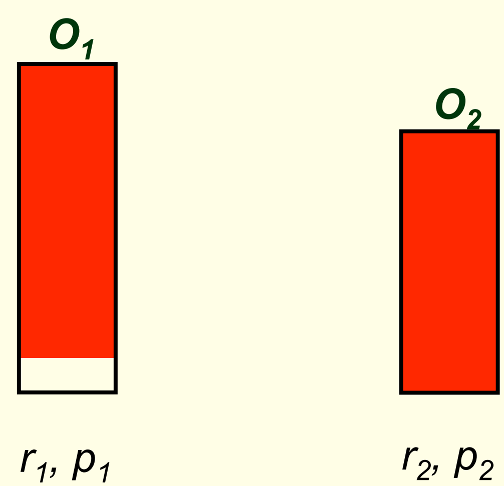
Start adding flow through (1, 2)



<u>Solution:</u>
During every time unit, send z tuples through (1, 2)

Or:

O_2 saturates, but O_1 doesn't



1. O_2 has saturated
2. No flow *<u>out of</u>* O_2

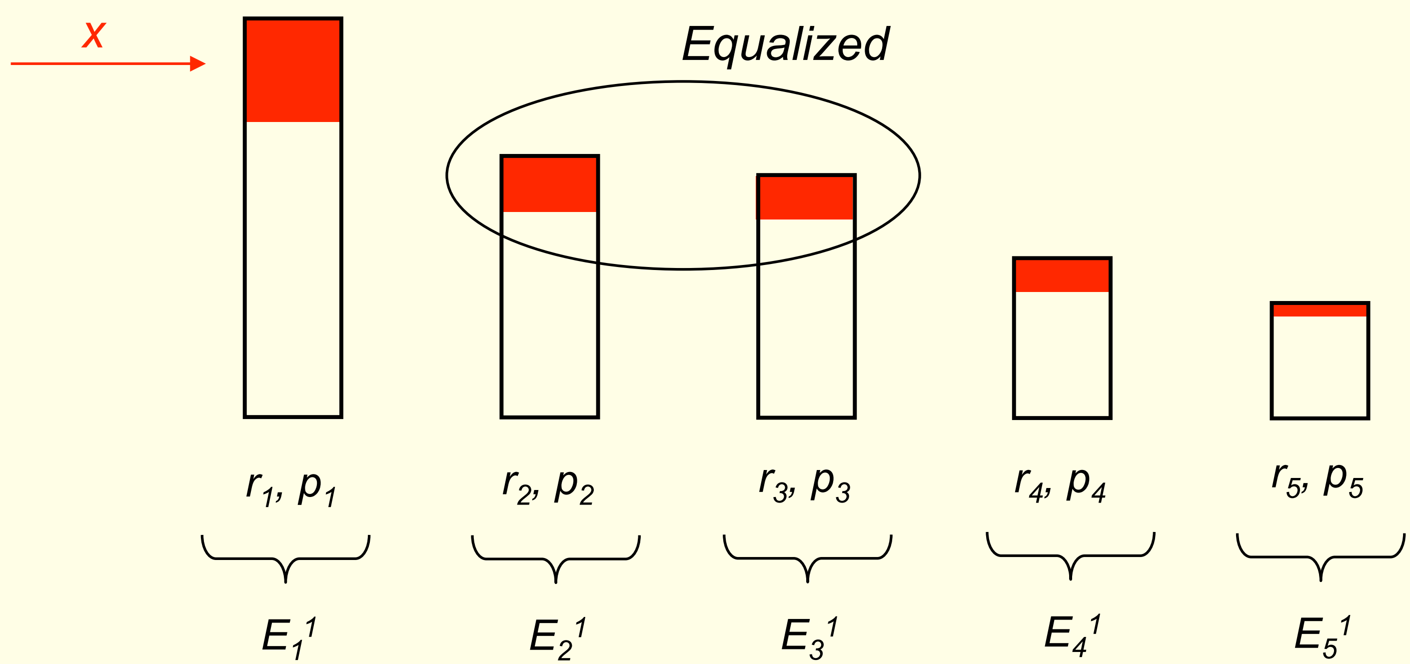
Optimal

O_2 is being applied *last* to all tuples.
No way to lighten its load more.

Unequal rates: General Case

- Create equivalence *groups* of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if *residual rates* equalize

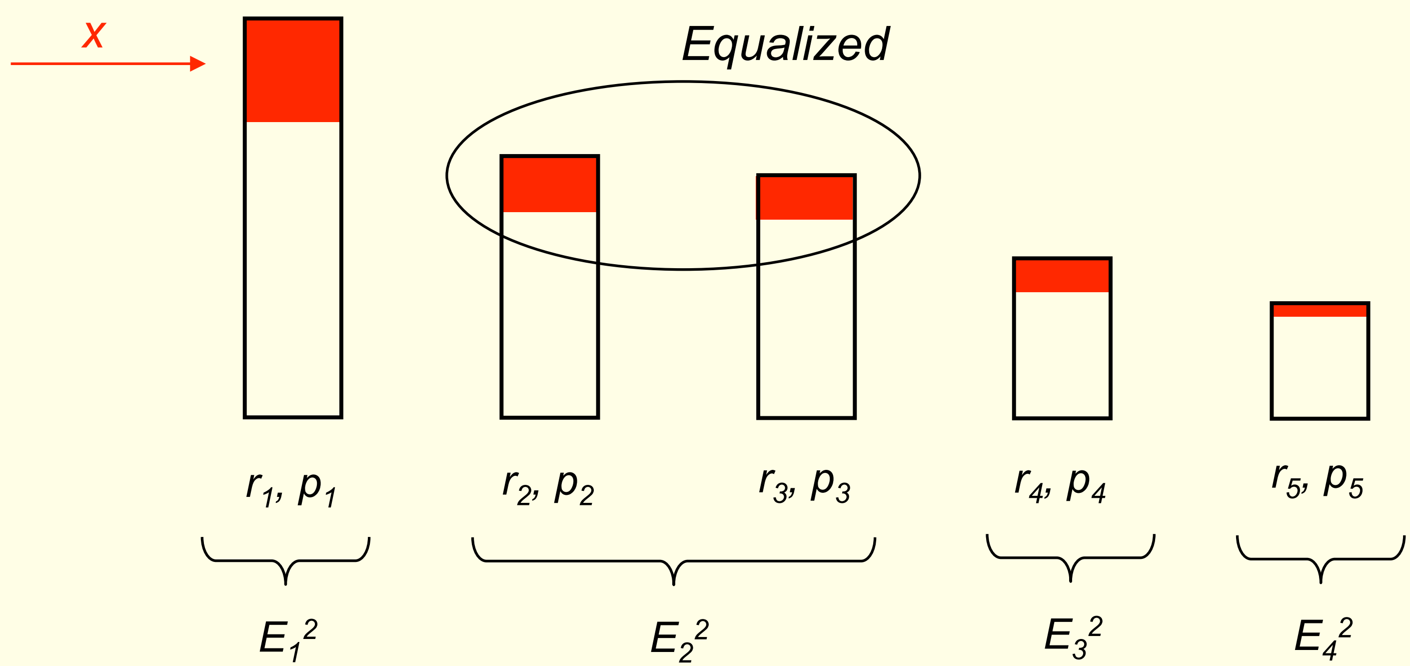
Partial solution:
1. send *x* tuples along *(1, 2, 3, 4, 5)*



Unequal rates: General Case

- Create equivalence *groups* of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if *residual rates* equalize

Partial solution:
1. send x tuples along *(1, 2, 3, 4, 5)*

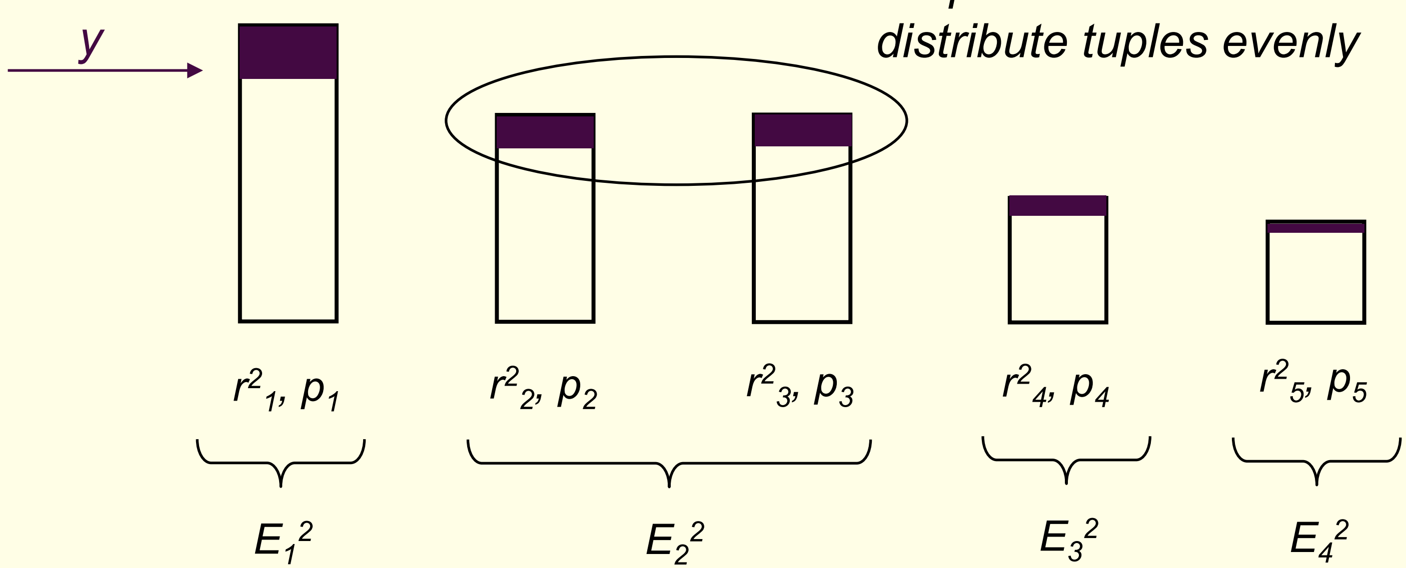


Unequal rates: General Case

- Create equivalence *groups* of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if *residual rates* equalize
- Recurse using residual rates

Partial solution:
1. send x tuples along $(1, 2, 3, 4, 5)$

Use equal-rates solution to distribute tuples evenly

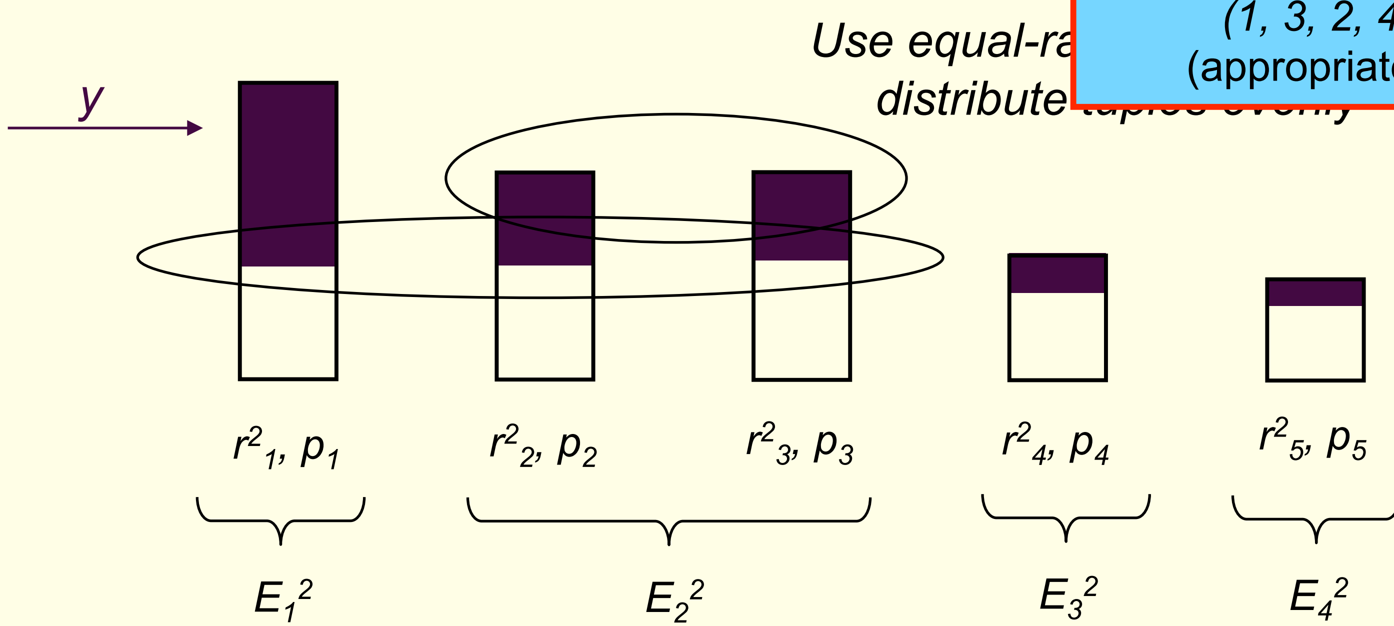


Unequal rates: General Case

- Create equivalence *groups* of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if *residual rates* equalize
- Recurse using residual rates

Partial solution:

1. send x tuples along *(1, 2, 3, 4, 5)*
2. send y tuples total along: *(1, 2, 3, 4, 5)* and *(1, 3, 2, 4, 5)* (appropriately divided)



Unequal rates: General Case

- Create equivalence *groups* of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if *residual rates* equalize
- Recurse using residual rates

- After at most *n* rounds
  - Either: All operators are equalized
    - Solve using equal rates solution
    - *Full saturation → Optimality*
  - Or: Rightmost equivalence group saturates
    - *No flow out of that group → Optimality*
- Running time: $O(n^2)$

Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
    - Equal rates, unequal selectivities
    - Unequal rates and selectivities
      - Two operators
      - General case
  - Routing and queuing issues
- Adversarial type, Single Tuple

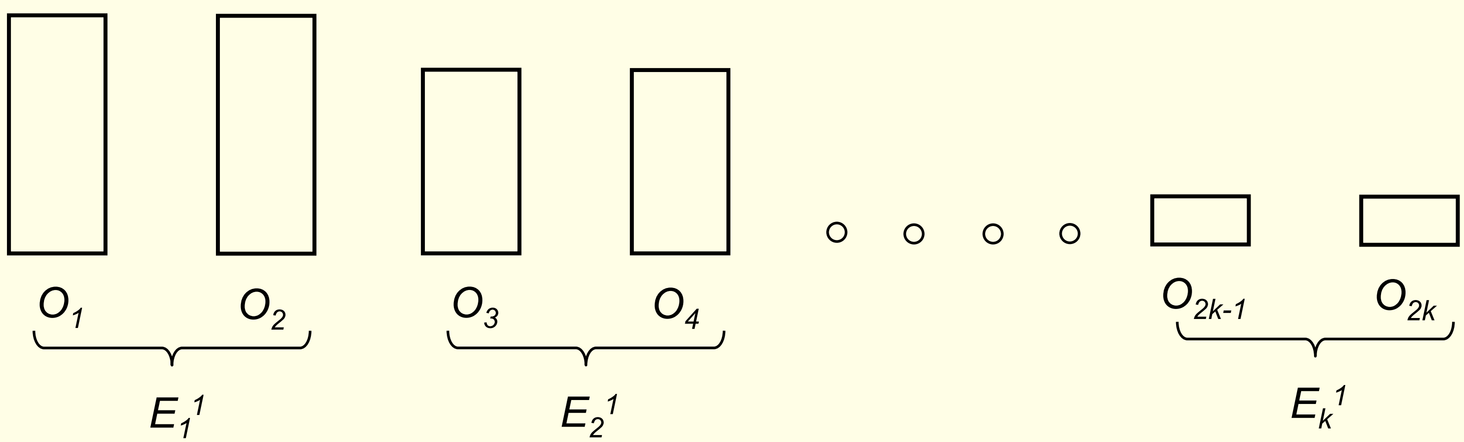
Routing

- Ideal form of the final solution:
  - *(π_i, $f_{\pi i}$): a list of permutations, and associated flows*
- Not feasible: The resulting solution is *not necessarily sparse*

- Can directly use the output of the algorithm (the equivalence groups and associated flows) for routing tuples

Positive flow assigned to 2^k permutations

(1 2) (3 4) … (2k-1 2k)



Queuing issues

- Rate limits only guaranteed in expectation

- [Kodialam'01]
  - If K is an optimal routing scheme with max throughput F, then, for any $F^* < F$, there is a routing scheme K^* with throughput F^* that obeys the rate limits

Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
- Adversarial type, Single Tuple

Problem 2: Adversarial, Single Tuple

- Given n predicates and their costs, $c_1, \ldots, c_n$
- For tuple t, let:

$$\textit{multiplicative regret}(t) = \frac{\textit{actual cost of processing } t}{\textit{minimum cost of processing } t}$$

- Adversary knows algorithm used, and controls input tuples
- *Goal: Minimize the expected multiplicative regret*

Problem 2: Adversarial, Single Tuple

- Can be reduced to a problem similar to max-throughput problem
  - Details in the paper
  - Same algorithms can be used

- Naïve solution
  - Order in the increasing order of costs

- Theorem: Naïve solution within a factor of 2

Conclusions

- Two pipelined filter ordering problems
  - Maximize throughput in a parallel scenario
  - Minimize multiplicative regret in an adversarial scenario
- Very simple and efficient algorithms
  - Using a *flow* formulation
- Interesting open problems
  - Correlated predicates
  - Robustness of algorithms
  - Routing tuples of multiple types together

Thank you !!

- Questions ?