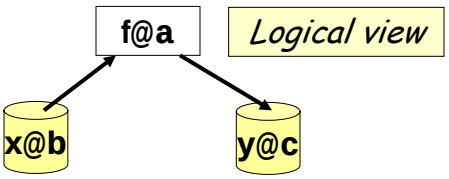
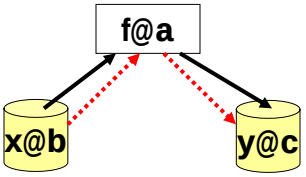
Back to Scientific Workflows

- Actor-oriented programming in Kepler
  - Actors: to do the acting
  - Director: to "factor out" the orchestration/control
  - BlackBox!?: to factor out the "flight recording"
    - e.g. by default leave a trace of certain ops in the black box…
  - BlueBox!?: to factor out the Grid resource allocation and scheduling
    - e.g. schedule an abstract workflow **without** specifying
      - on which host a job is run
      - which data transfer protocol is used between hosts
      - how jobs and data is shipped … (SHA)

Shipping and Handling Algebra (SHA)

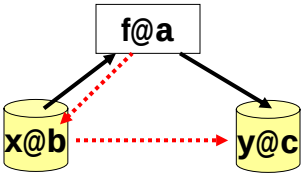




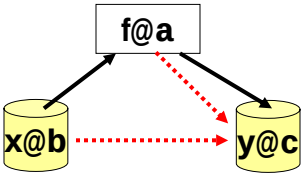
(1)

plan Y@C = F@A of X@B =

1. **[X@B to A, Y@A := F@A(X@A), Y@A to C]**
2. **[F@A => B, Y@B := F@B(X@B), Y@B to C]**
3. **[X@B to C, F@A => C, Y@C := F@C(X@C)]**



(2)



(3)

Physical view: SHA Plans

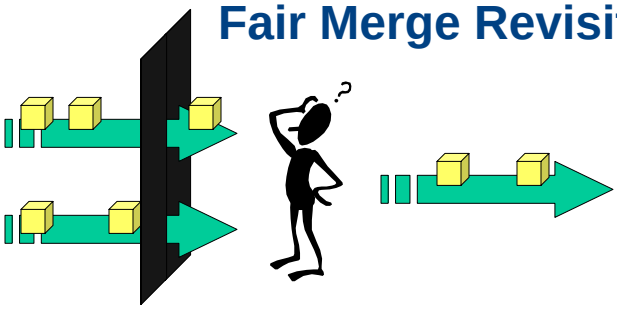
An (oversimplified) **Model of the Grid**

- *Hosts*: {h1, h2, h3, …}
- *Data@Hosts*: d1@{h_i}, d2@{h_j}, …
- *Functions@Hosts*: f1@{h_i}, f2@{h_j}, …

- Given**: data/workflow:**
- **… as a functional plan:** **[…; Y := f(X); Z := g(Y); …]**
- **… as a logic plan:** **[…; f(X,Y)∧g(Y,Z); …]**

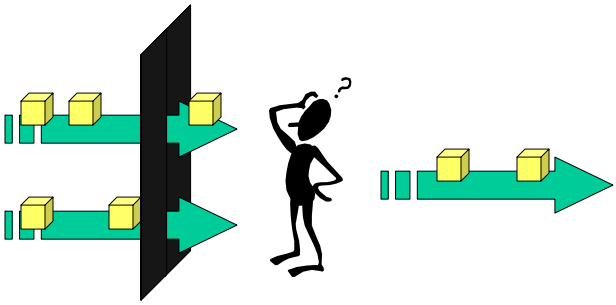
- Find *Host Assignment*: **d_i ➔ h_i , f_j ➔ h_j**
  - **for all d_i , f_j** … s.t. […; d3@h3 := f@h2(d1@h1), …] is a **valid** plan

Fair Merge Revisited



- Fairness:
  - don't let any channel ("assembly line") starve
- Determinism:
  - input sequence functionally determines output sequence

Process Network w/ “One Peek” (PN1P)

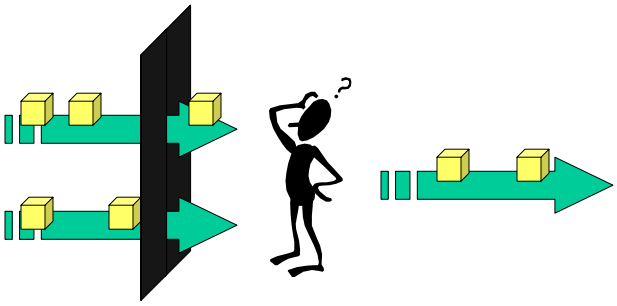


Fair, Non-Deterministic (FND):

1. L1[X], L2[Y] --> del(X), del(Y), out [X,Y]
2. L1[X], L2[] --> del(X), out [X]
3. L1[], L2[Y] --> del(Y), out [Y]
4. L1[], l2[] --> []

- FND is **non-deterministic** since the arrival order (which is outside of the model) determines the output of merge([x1,x2,...], [y1,y2,...])
- FND is **fair**, since even if the sequences have different lengths, they are served on a first-come, first-served basis.
- In particular, even if one line/channel produces tokens much faster than the other, a bounded buffer is sufficient (provided the merge actor itself works fast enough)

Process Network w/ “One Peek” (PN1P)

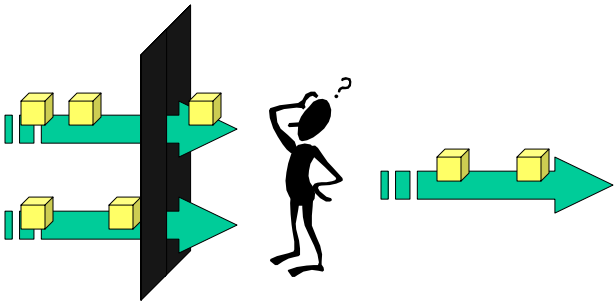


DUM (Deterministic, Unfair Merge):

1. L1[X] < L2[Y] --> del(X), out [X]
2. L1[X] > L2[Y] --> del(Y), out [Y]
3. L1[X] = L2[Y] --> del(X), del(Y), out [X,Y]

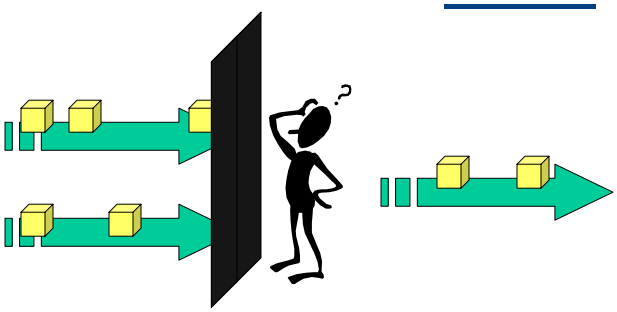
- Here we assume that the tokens X and Y have a built-in "serial number“ (an integer) and we know that each channel produces increasing sequences of serial numbers (with unknown gaps though). Note that case (3) can be avoided simply by making serial numbers unique.
- DUM is **deterministic** since for any two input sequences (including knowledge of each token's serial number), the output sequence is determined (increasing serial numbers).
- DUM is **unfair** since we must request one token each for the comparison of the merge step, which unfairly starves the longer sequence (say L2), while waiting for the (never appearing) "post-ultimate" token of the shorter sequence (say L1). That is, once we've consumed the last token of L1, we keep waiting indefinitely for a token that won't arrive.

Process Network w/ “One Peek” (PN1P)



- So it seems that in this setting we have a trade-off between determinism and fairness…
- NOTE: In practical settings, one will have a timeout associated with each channel, so that after not having seen a token after a sufficiently long time on some channel (say L1), it is "known" that no token will arrive on L1, thereby allowing the actor to process the tokens from L2. If the "time-out assumption" is valid (no token can arrive after the time-out), then this yields a fair (and still deterministic) merge actor.
- However, if the time-out assumption is violated (which even happens in practice), then the actor becomes non-deterministic again, since the merge actor can prematurely pick the "wrong" token (the one with the higher serial number), assuming nothing will arrive on the presumably “dead” channel

Process Network without Peek (PN)



Question/Exercise:

What happens to the previous cases if you cannot “peek” behind the black wall (or similarly, if as in the case of Ptolemy/PN you always get “true” but are “put to sleep” when you are waiting for the requested token that’s not (yet/ever) there?