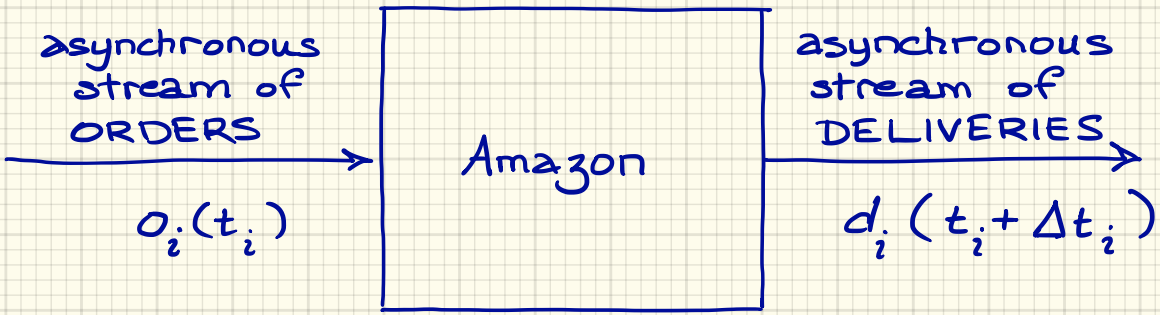
Time Warp 2018

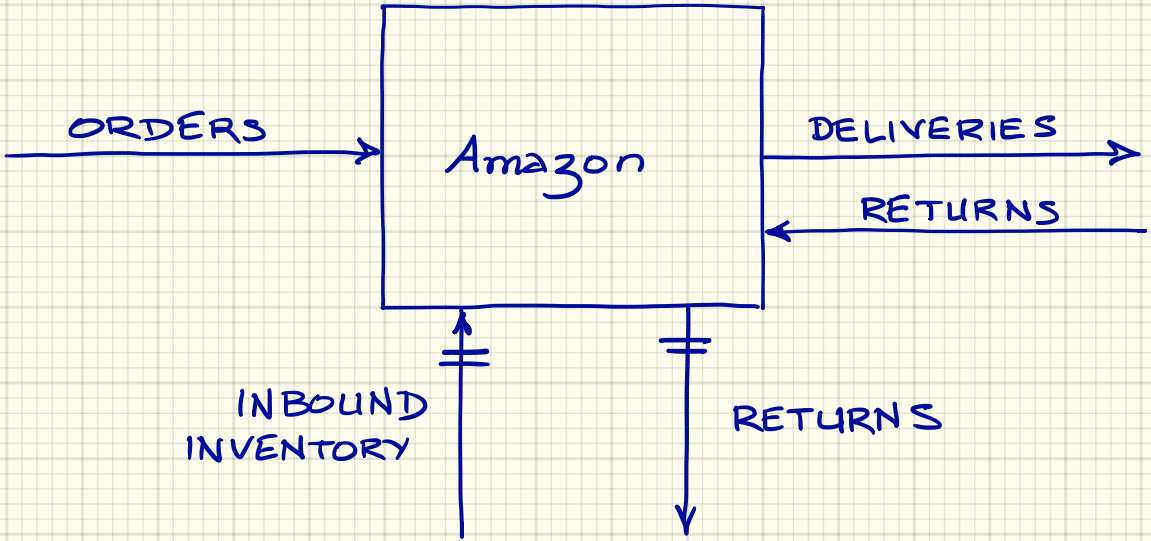
Distributed Discrete-Event Simulation

Amazon is a Function from orders to deliveries

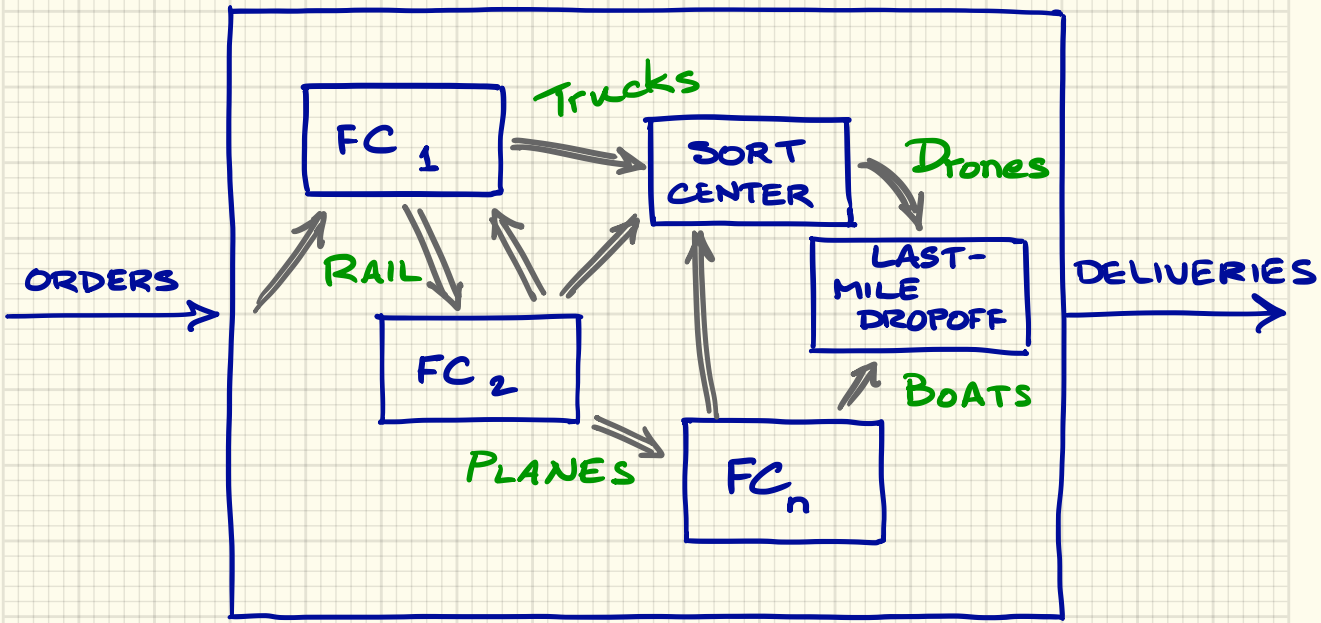


The function itself depends on time due to inventory & returns

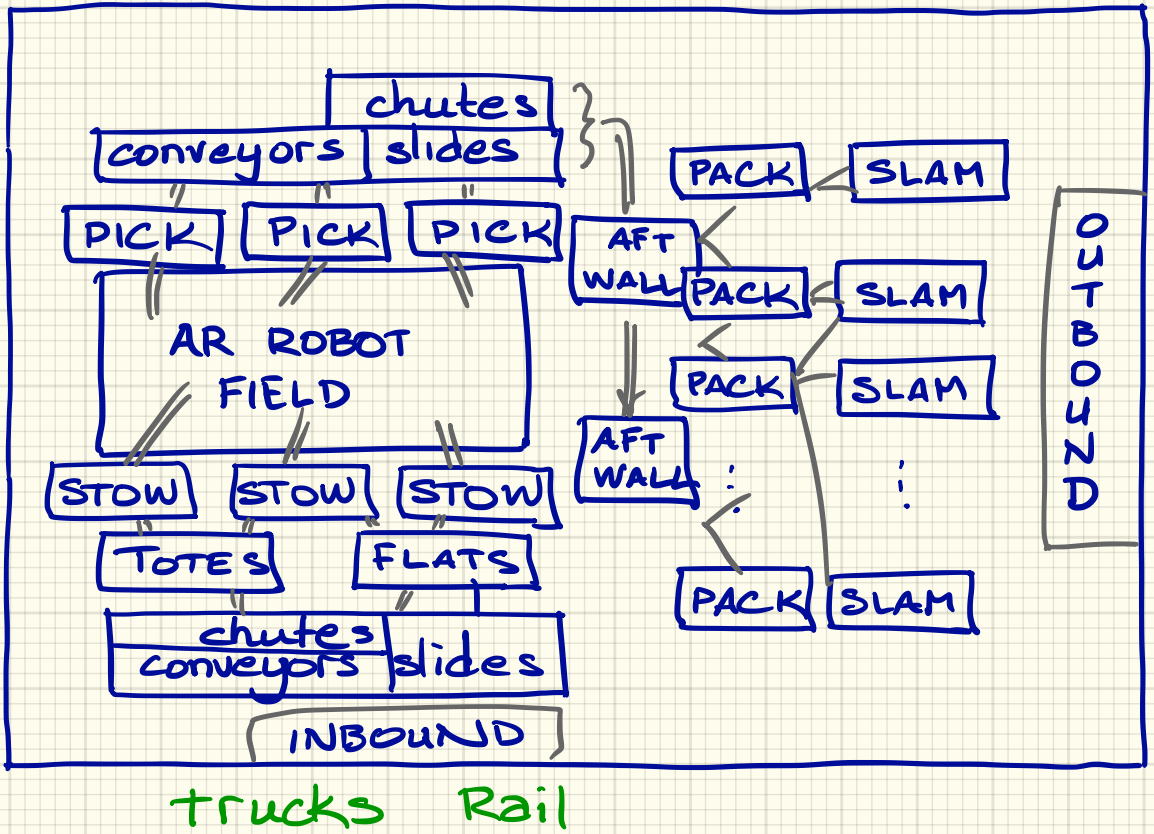
we'll ignore that for now



One Level Down



One More Level Down



Questions:

Billion-dollar decisions

- geography & velocity of demand

  ⇒ how many FC's? where?

  ⇒ how many conveyors? robots? in the FC, sort center, last-mile station

  ⇒ how many trucks, planes, containers? where? when?

- FINANCE is a whole 'nuther dimension ...

  ⇒ how much? borrow? sell stock? when?

maybe even compose with detailed mech sims of individual robots & vehicles

Analytic solutions out of the question

Queuing model with fixed flows?

OR

Discrete-Event Simulation

- set up components & flows
- do "what-if's" (COUNTERFACTUALS)

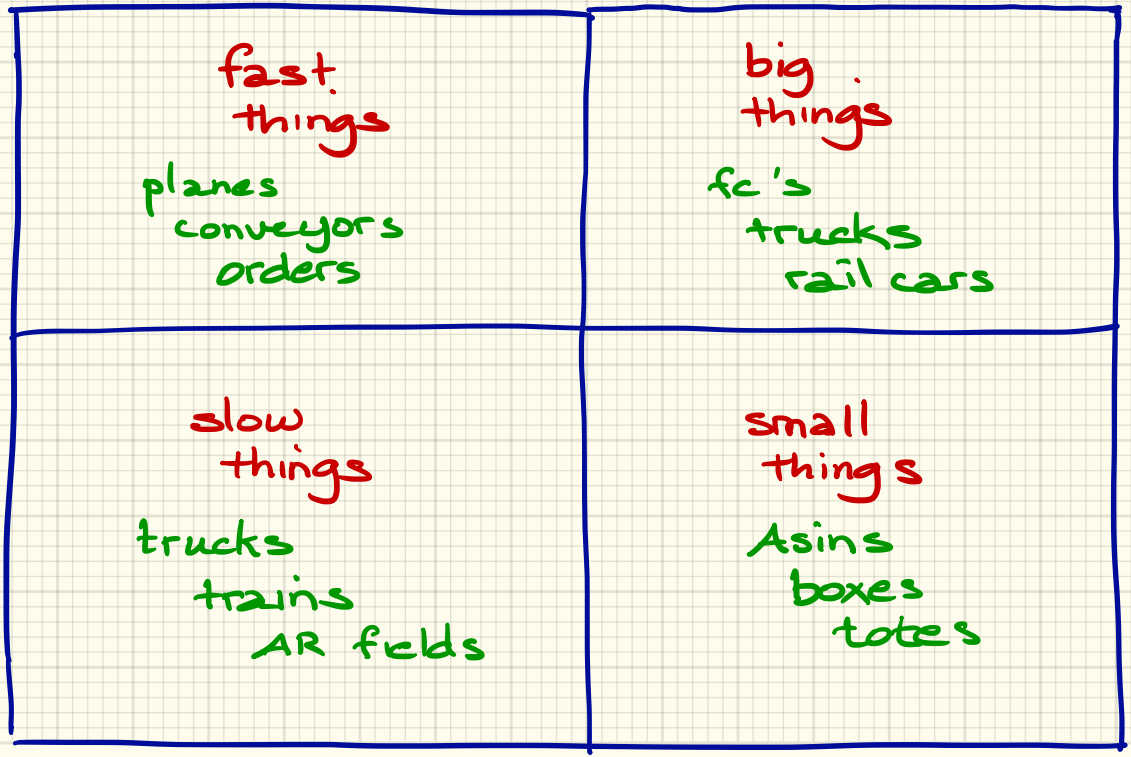
— put a big FC in {Kansas, Texas, Alaska}

— run some sims, see what happens

— put {more/fewer} {faster/slower}
{conveyors / transports}
see what happens

IDEAL FOR TIME WARP

Millions of logical processes
irregular space & time structure
runs much too slowly NEED SPEEDUP



Time-stepped sim: step every process at the rate of the slowest one

⇒ most processes idle most of the time

Optimistic sim (aka speculative): processes run with best-guess inputs, rollback & cancel incorrect outputs when necessary

Time-stepped sim: step every process at the rate of the slowest one

⇒ most processes idle most of the time

Optimistic sim (aka speculative): processes run with best-guess inputs, rollback & cancel incorrect outputs when necessary

CAN WE DO THIS DISTRIBUTED, IN PARALLEL, GET SPEEDUP FROM PARALLEL EXECUTION?

Synchronous, Conservative, parallel simulation has been studied extensively:

Never do incorrect computation

Block & wait or deadlock - and break

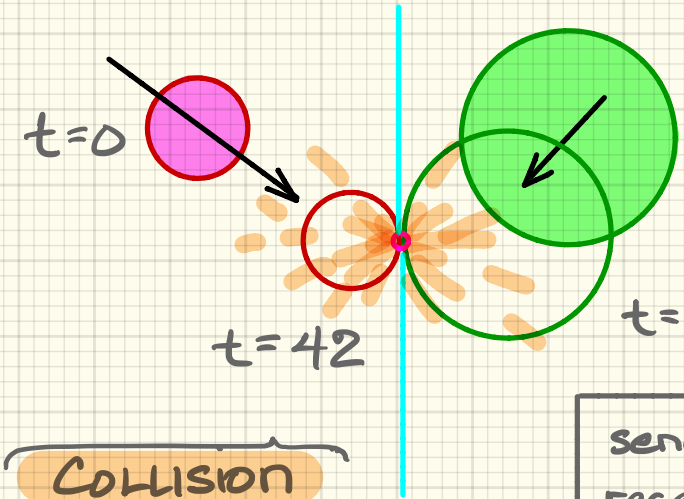
Requires order-preserving queues

Lacks essential symmetries of TW

Requires static graph of comm. channels

Slower than TW for irregular problems

(faster than TW for some network sims)



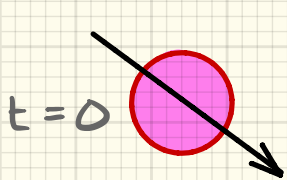
DETAILS OF THE DEMO

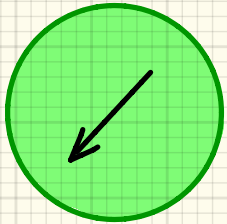
PROGRAMMING MODEL

COLLISION PREDICTED

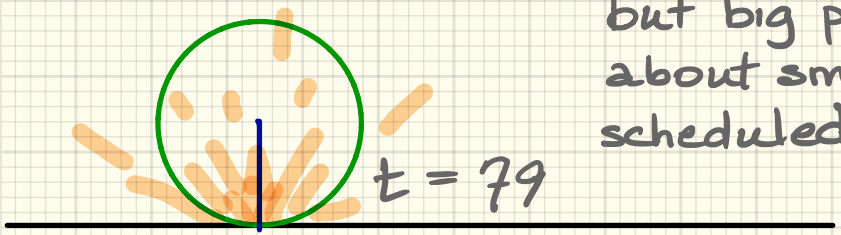
- FOR time 42
- AT time 0

sender: small puck
receiver: big puck
send time: 0
receive time: 42
body: your new center & velocity will be ...



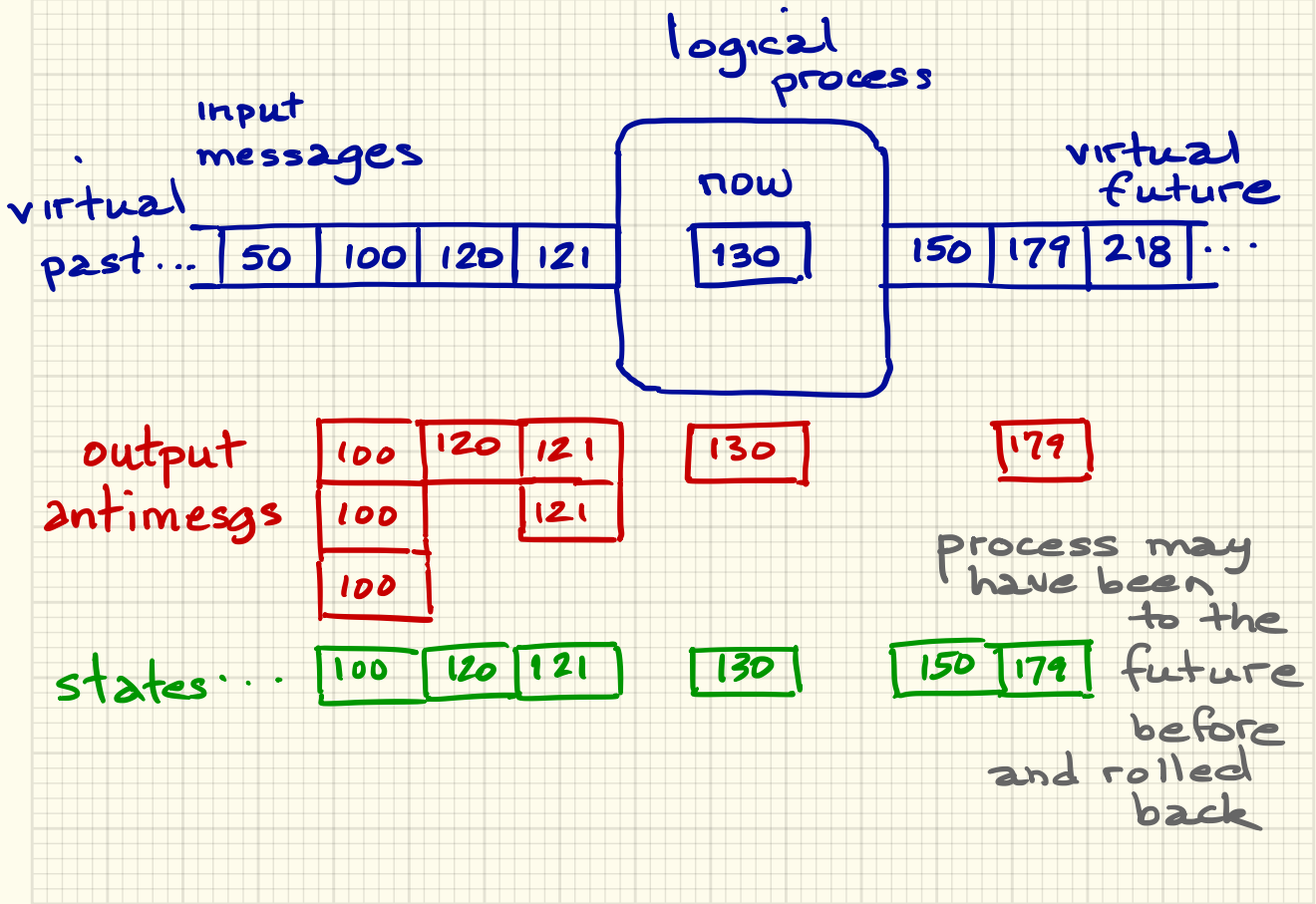


But, but...



but big puck didn't know about small & already scheduled a later collision with the wall!

what to do? : small causes big to roll back & cancel the incorrect wall collision by sending antimessage to self. SYSTEM DOES THIS AUTOMATICALLY — NO USER CODE NEEDED


logical process
input messages
virtual past...
50
100
120
121
now
130
virtual future
150
179
218
...
output antimesgs
100
120
121
130
179
100
121
100
process may have been to the future before and rolled back
states...
100
120
121
130
150
179

Side Note:

All forms of speculative computation at all levels can benefit from Time Warp

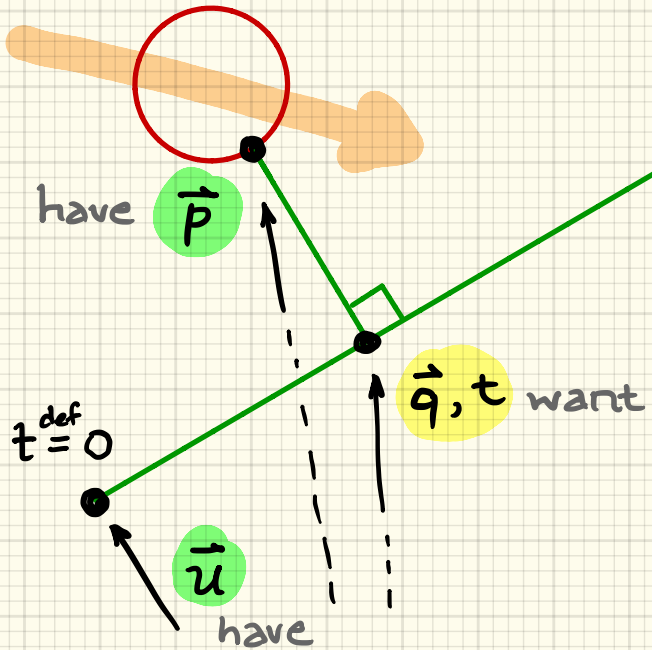
{Synchronization by rollback alone
Cancellation by antimesage alone}

These are fundamental concepts

⇒ Apply them at CPU & GPU levels

Basic physics calcs: wall vs. puck

COLLINEAR POINT AND PARAMETER



$t \stackrel{\text{def}}{=} 1$ have $\vec{v}$

$$t = \frac{(\vec{p} - \vec{u}) \cdot (\vec{v} - \vec{u})}{\| \vec{v} - \vec{u} \|^2}$$

$$\vec{q} = \vec{u} + t(\vec{v} - \vec{u})$$

1. Find contact normal vector

$$\frac{\vec{q}-\vec{c}}{|\vec{q}-\vec{c}|} \overset{\text{def}}{=} \hat{n} \qquad \vec{c} \overset{\text{def}}{=} \text{center of puck}$$

2. Find tangent vector

$$\begin{bmatrix}0\\0\\1\end{bmatrix} \times \begin{bmatrix}\hat{n}_x\\\hat{n}_y\\0\end{bmatrix} = \begin{bmatrix}\hat{n}_y\\-\hat{n}_x\\0\end{bmatrix} \overset{\text{def}}{=} \hat{t}$$

3. Resolve velocity

$$\vec{v} = (\vec{v}\cdot\hat{n})\,\hat{n} + (\vec{v}\cdot\hat{t})\,\hat{t}$$

4. Reverse normal component:

$$\vec{v}' = -(\vec{v}\cdot\hat{n}) + (\vec{v}\cdot\hat{t})\,\hat{t}$$

Wall is an infinite source of momentum

$\Leftarrow$ because its mass is infinite

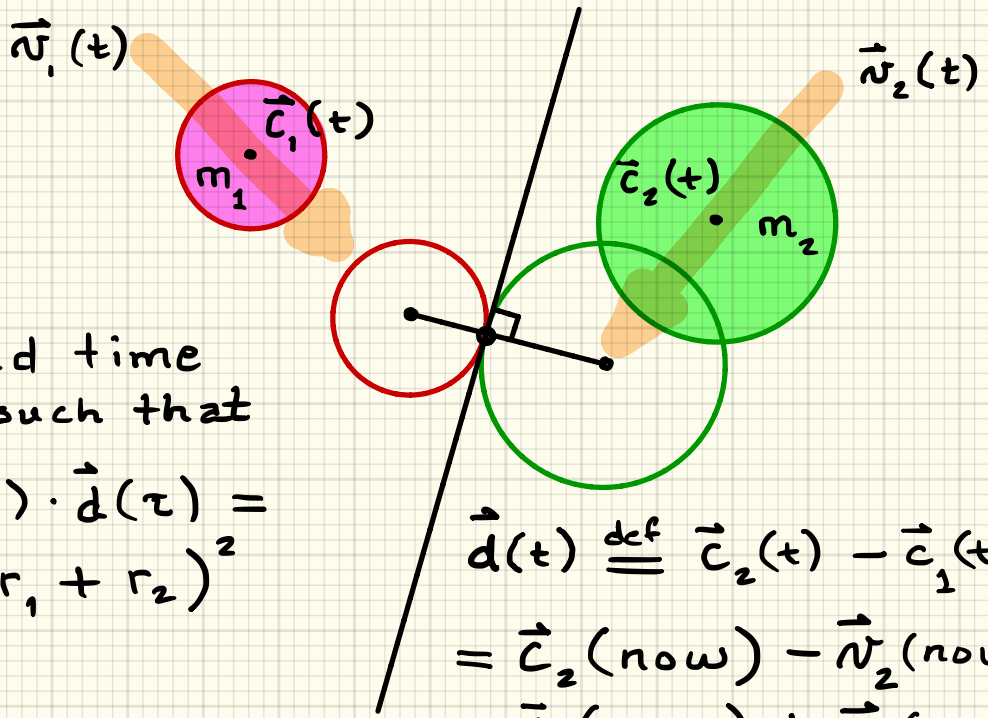
Puck energy is conserved

$$\frac{1}{2} m \vec{v} \cdot \vec{v} = \frac{1}{2} m \vec{v}' \cdot \vec{v}'$$

(check this at run time)

$\Leftarrow$ collision is perfectly elastic

Basic Physics calcs : puck vs. puck



find time τ such that

$$\vec{d}(\tau) \cdot \vec{d}(\tau) = (r_1 + r_2)^2$$

$$\vec{d}(t) \stackrel{\text{def}}{=} \vec{c}_2(t) - \vec{c}_1(t)$$

$$= \vec{c}_2(\text{now}) - \vec{v}_2(\text{now})\,t - \vec{c}_1(\text{now}) + \vec{v}_1(\text{now})\,t$$

$\Leftrightarrow$ find τ the zero of

you do the arithmetic

$$\|\vec{v}\|^2 t^2 - 2\,\vec{d}(now)\cdot\vec{v}\,t + \|\vec{d}(now)\|^2 - d_1^2$$

$\vec{v} \stackrel{def}{=} \vec{v}_2(now) - \vec{v}_1(now)$

$d_1 \stackrel{def}{=} r_1 + r_2$

$a \stackrel{def}{=} \|\vec{v}\|^2$

$b \stackrel{def}{=} -2\,\vec{d}(now)\cdot\vec{v}$

$c \stackrel{def}{=} \|\vec{d}(now)\| - d_1^2$

$$\tau = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

collision at smallest positive real root if $\exists$

Conservation of momentum & energy implies

(you do the arithmetic)

$$\vec{v}_{1n}' = \left[(m_1 - m_2)\vec{v}_{1n} + 2m_2\vec{v}_{2n}\right]/M$$

$$\vec{v}_{2n}' = \left[(m_2 - m_1)\vec{v}_{2n} + 2m_1\vec{v}_{1n}\right]/M$$

where subscript "n" means "normal component." Tangential components not changed. $M \stackrel{\text{def}}{=} m_1 + m_2$

Check conservation every ~~step~~ event:

$$\frac{1}{2} m_1 \vec{v}_1'^{\,2} + \frac{1}{2} m_2 \vec{v}_2'^{\,2} = \frac{1}{2} m_1 \vec{v}_1^{\,2} + \frac{1}{2} m_2 \vec{v}_2^{\,2}$$ (scalar equation)

$$m_1 \vec{v}_1' + m_2 \vec{v}_2' = m_1 \vec{v}_1 + m_2 \vec{v}_2^{\,2}$$ (vector equation)

Extend the system with

1. rotation (spin)
2. puck-table friction
3. puck-puck friction
4. inelasticity
5. jumps off the table

} Can still be solved exactly

i.e., ∃ pool-ball discrete-event sim

Invitation 1 (there are more)

Improve my demo app:

- desymmetrize the pucks: each schedules each collision for the other. That's double work for puck-puck collisions
- add more pucks
- add more instrumentation, strengthen the verification, find bugs
- go to 3 dims & friction: pucks → pool

Invitation 2:

make this run on multiple processors
(hard, but not too hard. Try MPI. You will have to implement lots more of TW)

Measure additional speedup due to parallism

(my one-processor demo only shows speedup from sequential prediction, though it also shows rollback & cancellation)

Invitation 3:

parallelize the table

let sectors coordinate ownership & collisions

(very hard: a puck/ball can be in as many as four sectors at once).

Add many more pucks

INVITATION 4:

make this a full O.S.

(open problem; hasn't been done since '84)

all other known impls are on TOP of other host O.S.s (ROSS, GOTW)

do it on a raw VM

Why is this good? ...

	Ordinary OS	TWOS
scheduling	round-robin, ...	lowest lvt first
synchronization	lock, semaphore mutex, pipe, ...	rollback
flowcontrol	throttling, windowing	cancelback
commitment	transactions	GVT
memory mgt	garbage collection, page fault	fossil collection (also GVT) rollback (time fault)
load mgt	balancing	process migration by LVT

Every OS function has a VT analog.

If TW is NOT an OS, all overhead is duplicated.

Invitation 5:

distributed fault tolerance
MIX TW with PAXOS/RAFT ?
Never been studied or tried AFAIK

Virtual Time will be a reliable guide
Multiple copies of a process can execute
in parallel with arbitrary skew in VT
Rollback & Cancellation ensur correctness
Time Warp is inherently Byzantine!

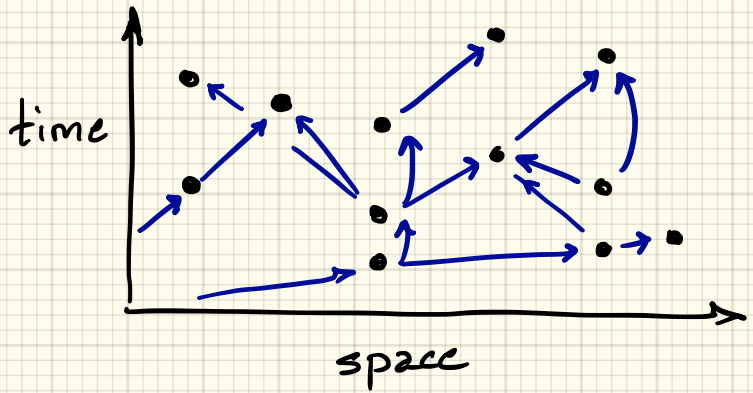
INVITATION 6:

Full space-time programming model

Events as points in spacetime

Causality emergent rather than forced

Function calls have space AND time loci



INVITATION 7:

SIMULATE AMAZON! on AWS of course!

run sims in seconds instead of weeks!

six-pager ?

AMZN on AMZN

WRAP-UP

- TW is best known method for parallel, distributed discrete-event simulation
- Speedups of >1 M have been demonstated
- It's been implemented many times, but not enough times as a full OS
- There are many juicy research problems
- Its symmetries often make the impossible relatively easy