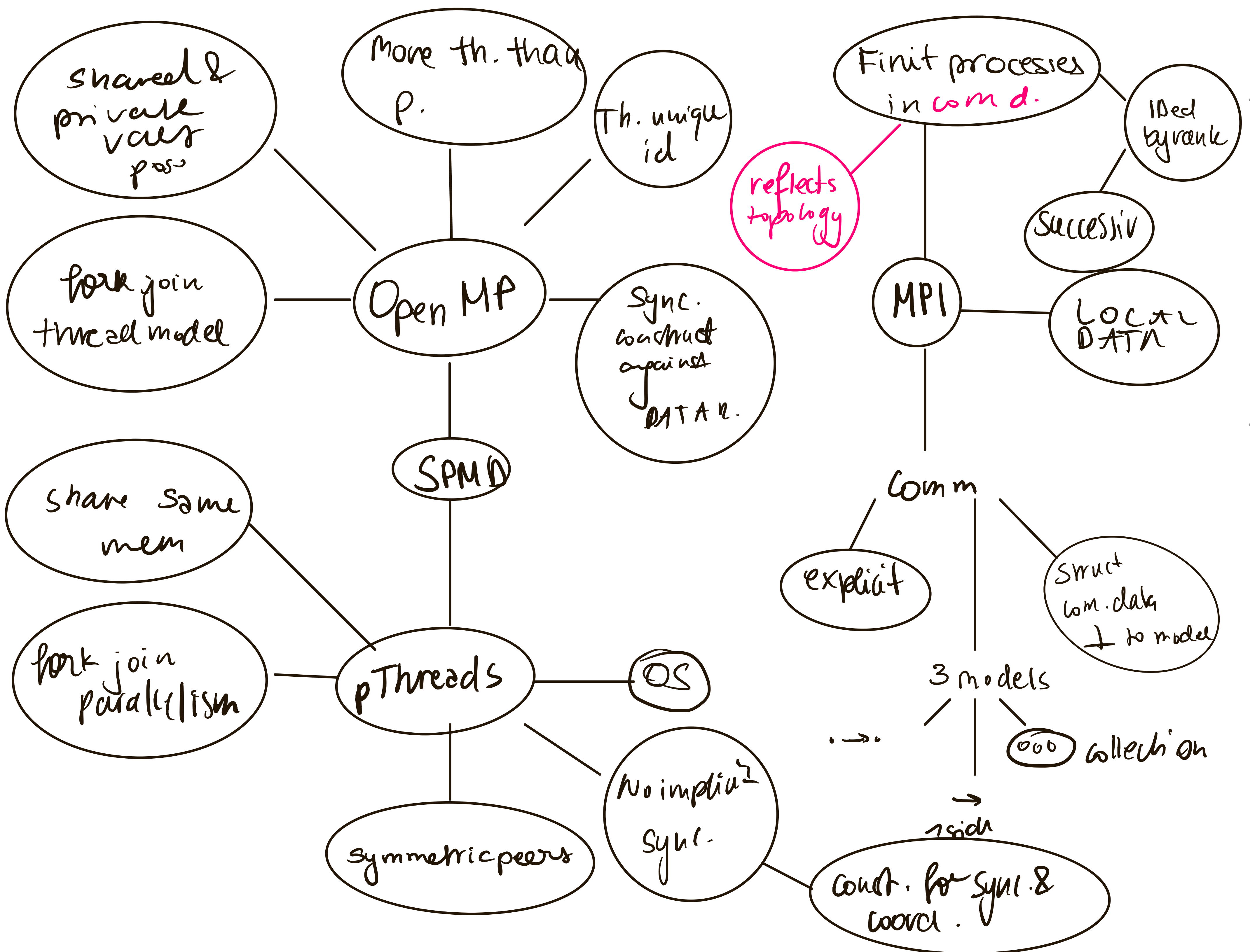
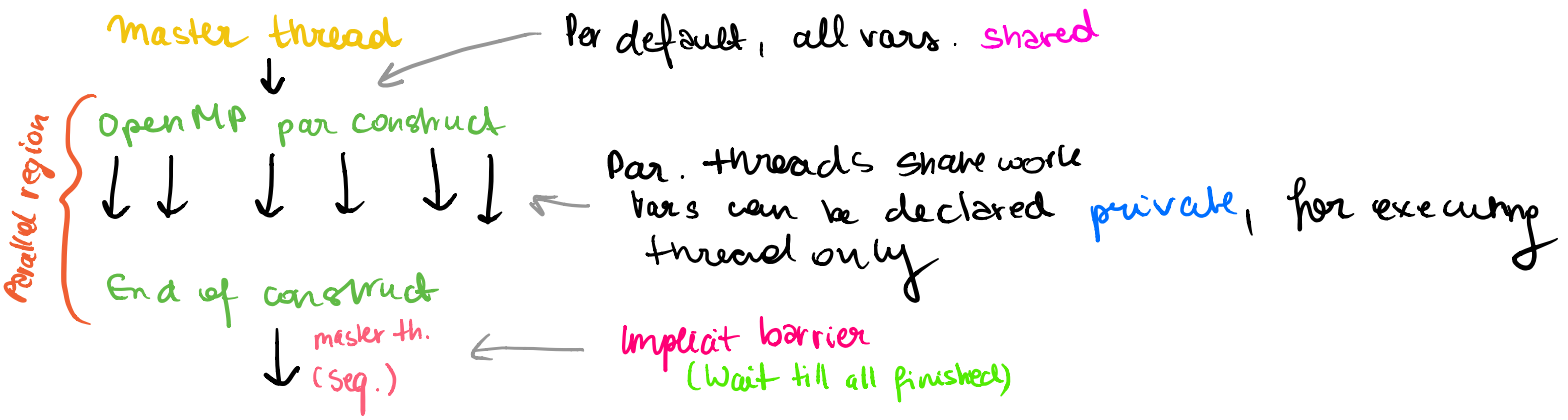
	OPEN MP	MSG PASSING MODEL	MPI
MODEL	• More threads than processes possible • Each thread unique id • Sync. constructs to prevent DATARACE • Shared & private vars poss • SPMD • Fork-join thread model thread forks & creates threads when done: threads join thread 	• Finit processes with LM • Expl. Comm • No implicit sync. → only comm. • Nothing shared	• Finite processes (more than 1 poss) in comm. domain • Each process identified by rank in domain • Ranks successive $(0, \dots, p-1)$ • LOCAL DATA • EXPLICIT COMM → Reliable & ordered • Structure of com. data ⊥ to node/model • Comm. d. reflect physical topology • No com. cost model • 3 basic comm models •→• Point-to-Point (diff. modes, local & ~~local~~ compl. semantics) → 1-sided (diff sync., local compl. semantics) ooo collective ops (~~local~~ compl. semantics)
	pThreads Fork-join parallelism → can spawn new threads symmetric peers → threads can wait for others to complete SPMD scheduled by OS No implicit sync. → threads are independent Share same memory Construct for sync. & coordination → MUTEX → R/W Locks → Condition vars		•→• Point-to-Point MPI_Send MPI_Recv } MPI sendrecv deadlock free reliable non-overtaking ordered MPI_Recv MPI_Send → 1 sided MPI_get (MPI_put) origin → target active → passiv Collective MPI_Barrier Bcast (root) Scatter(v) (All) gather(v) (All) Reduce MPI_All to All (Ex) Scan



shared & private vars pos
More th. than p.
Th. unique id
Open MP
fork join thread model
Sync. construct against DATA R.
SPMD
share same mem
fork join parallelism
pThreads
OS
No implicit sync.
symmetricpeers
Finit processes in com d.
reflects topology
IDed by rank
Successiv
MPI
LOCAL DATA
Comm
explicit
Struct com. data ⊥ to model
3 models
collection
1 side
Cond. for sync. & coord.

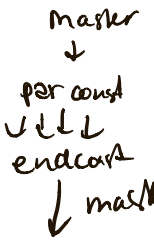
OPENMP



Data transfer between s. & p. copies var. is transparent, impl. & can be controlled

Parallel region

```
#pragma omp parallel
{
  ...
}
```



← "memory model"

→ Memory is consistent (state) by end of const.

↳ updates to shared var visible

Threads asynchron

MPI Equivalents

All collective func. blocking & not sync

MPI_Send
MPI_Recv
MPI_request
MPI_Irecv
MPI_Send
MPI_Wait

} = MPI_Sendrecv

MPI_Isend, MPI_Wait } = MPI_Send

MPI_Recv, MPI_Sendrec, MPI_Send } = MPI_Gather

MPI_gather
MPI_Bcast
MPI_Bcast
MPI_Bcast
memcpy

} = MPI_AllGather (= p · Bcast)

1

blocking

not sync

Can do:

i ... root

Bcast i
Bcast j

Bcast } i
Gather }

Bcast } k

Gather } j

Bcast } i
Recv(j) }

Isend(i) }
Bcast } j

•→• POINT-TO-POINT

Reliable
Deadlock-free
Non-overtaking (ordered)

MPI_Send → MPI_Recv
MPI_Recv ← MPI_Send
} Blocking

MPI_Isend (IMMEDIATE, LOCAL) — MPI_Irecv
NON Blocking

Successful: $0 \leq rank < size$

1. Sender specifies valid rank within comm. & valid (allocated) buffer
2. Receive with matching source rank & tag is eventually posted on same comm.
3. Sent data $<=$ to received amount
4. Type sig of sent & received data matches

↯ ERROR: Caught by MPI_Send: error

↳ IF NOT: DEADLOCK
↳ OTHERWISE: MPI_ERR_TRUNCATE
|| memory corruption

MPI_Send

- Short msg: Buffered at Sender / receiver → Processed later. MPI_Send can return IMMEDIATELY
- Medium msg: may be buffered locally. Sent when rec. has been posted. MPI_Send can return fast
- Large msg: participation of receiving process needed. MPI_Send cannot return before MPI_Recv becomes active

COST: $\alpha + \beta m$
- Latency: Time to get 1st data elem from Source to dest. processor
- bandwidth: How much data/time unit (s) can be transferred from Source to dest. processor

→ 1-sided

comm. between 2 processes
ONLY 1 Explicitly involved with comm

MPI_Get →
MPI_Put ←
} Non-Blocking

only origin Initiates comm & provides all args
PUT (origin → target, target not involved)
GET
MPI_Accumulate ( )

origin puts/gets data from stand. MPI_buffer (buf, count, type)

local completion semantics

No guarantee that data have arrived before Sync. operation

Comm. window

Collective Operations

All processes involved

Different for ↗ •→• & →

Same root → If diff. roots DEADLOCK

#sent = #received

Matching type signatures (pairwise same size)

Difficult to impl: perf. correct. safety

COLLECTIVE OPERATIONS

MPI_Bcast: Data from root → a

MPI_Scatter: Indiv. data root → all

MPI_Gather: Indiv. data all → root

not evenly +v

MPI_Allgall: Indiv. all → all

MPI_Reduce: Apply ass. function(+,*,...) to data from each process
res at root

MPI_Allreduce: res to all

MPI_Reduce_Scatter: res scattered/distr.

Pi (non-root)
R(sbuf, rbuf, ... root, comm);
Pj (root)
R(sbuf, rbuf, root, comm);
only sig. for root

MPI_Scan: Prefix sum (reduction)

MPI_Exscan: value of proces not used in reduction

MPI_Barrier: Sync. (waits until have reached routine)