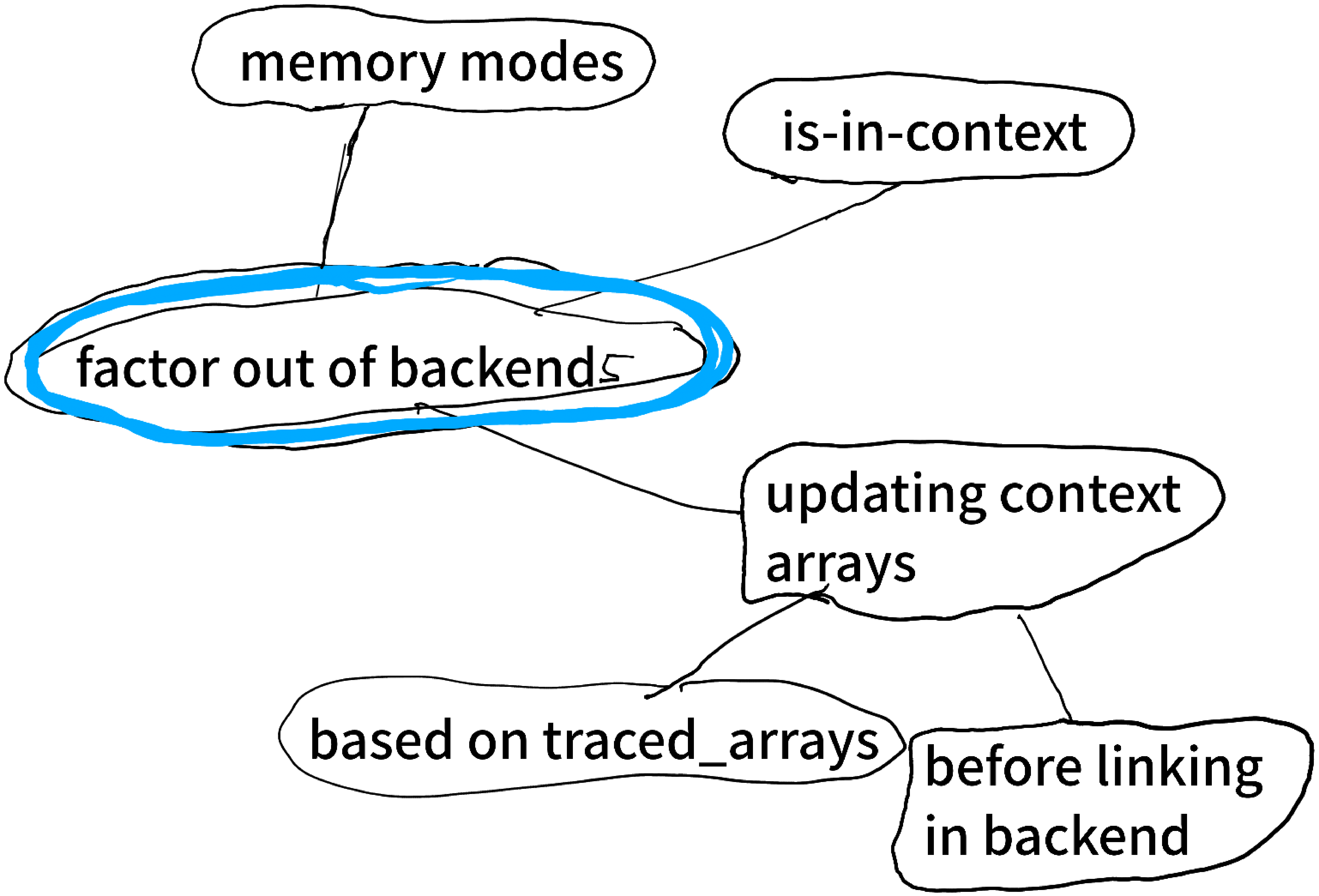
2024.11.07



memory modes

is-in-context

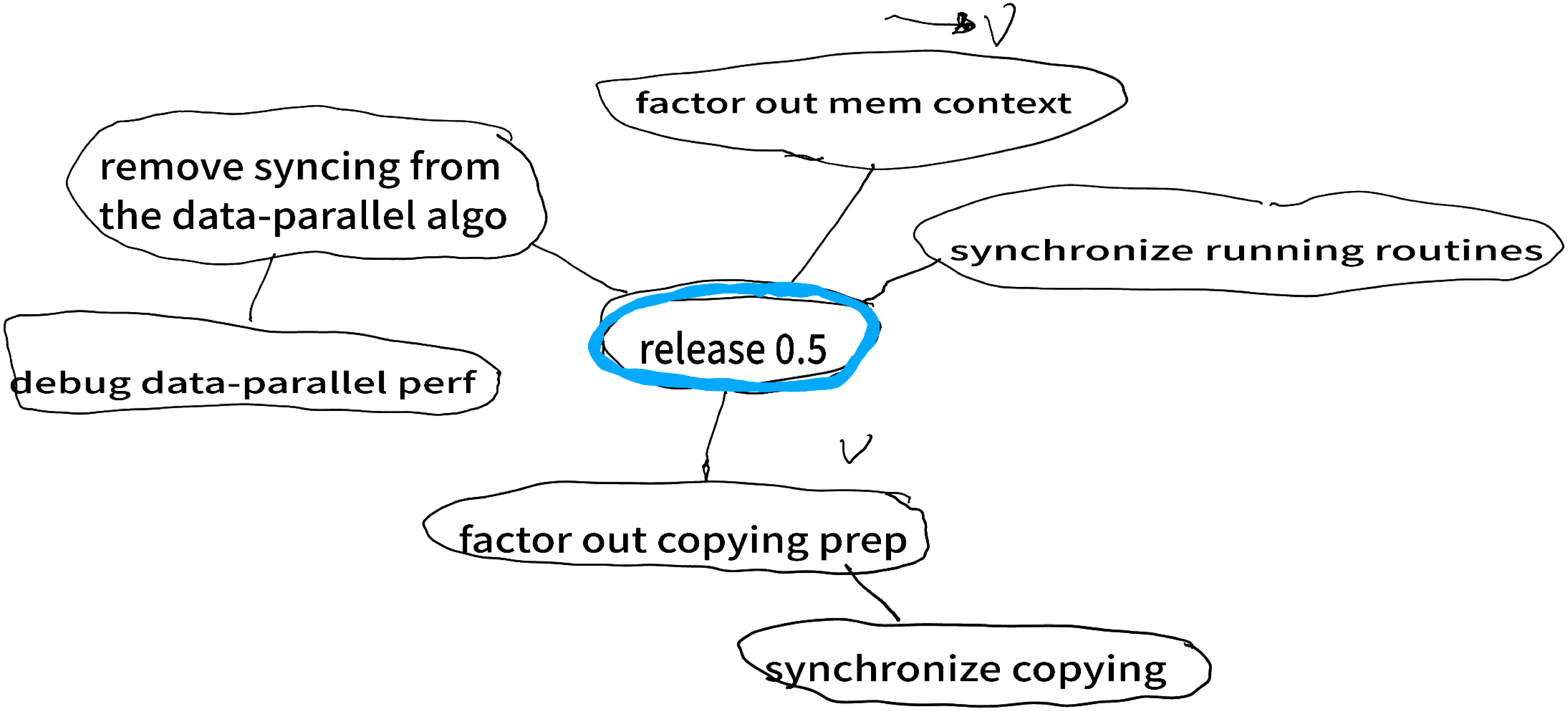
factor out of backends

updating context arrays

based on traced_arrays

before linking in backend

2024.11.08



release 0.5

factor out mem context → ✓

synchronize running routines

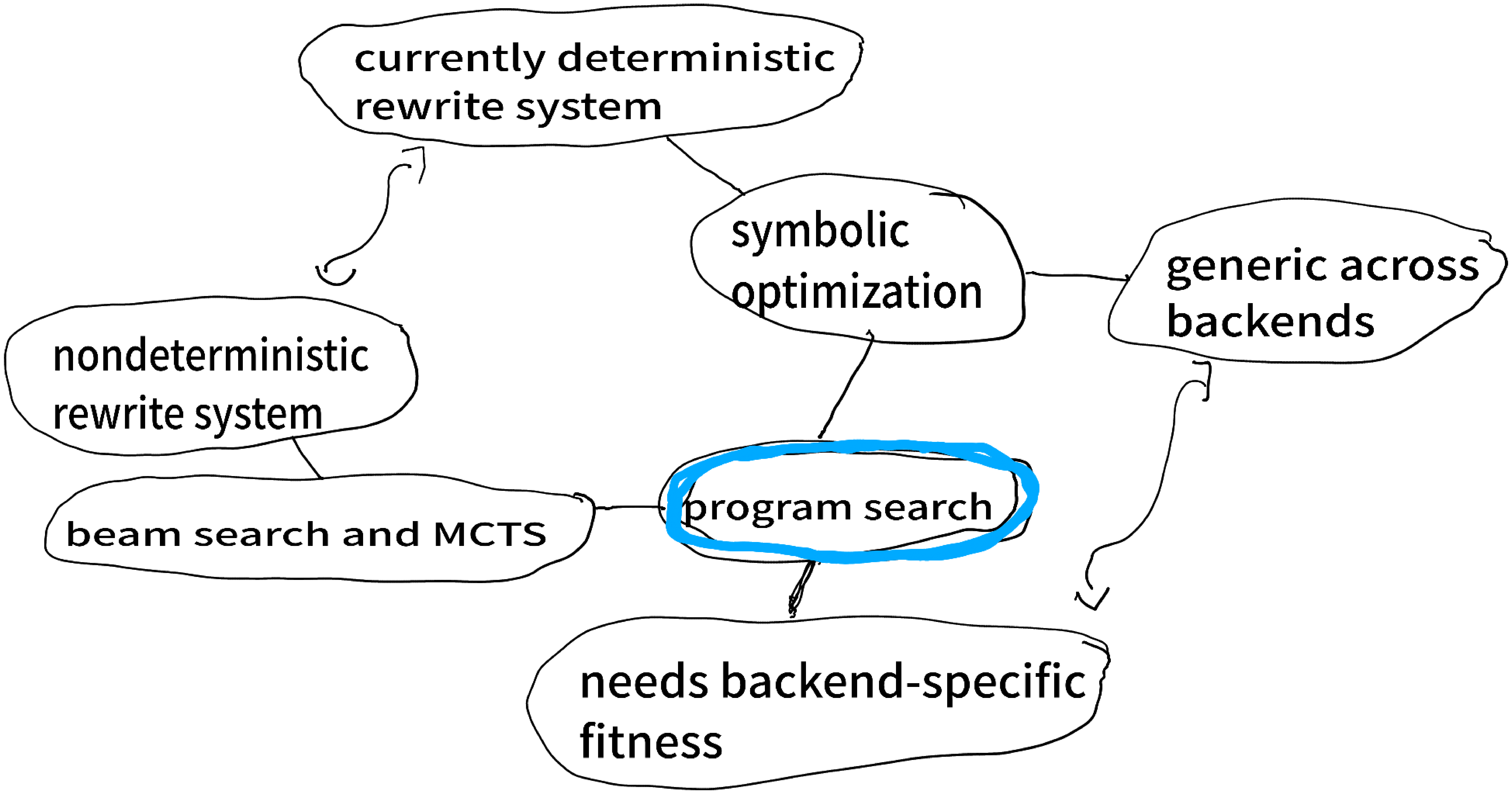
remove syncing from the data-parallel algo

debug data-parallel perf

factor out copying prep ✓

synchronize copying

2024.11.09



currently deterministic rewrite system

nondeterministic rewrite system

symbolic optimization

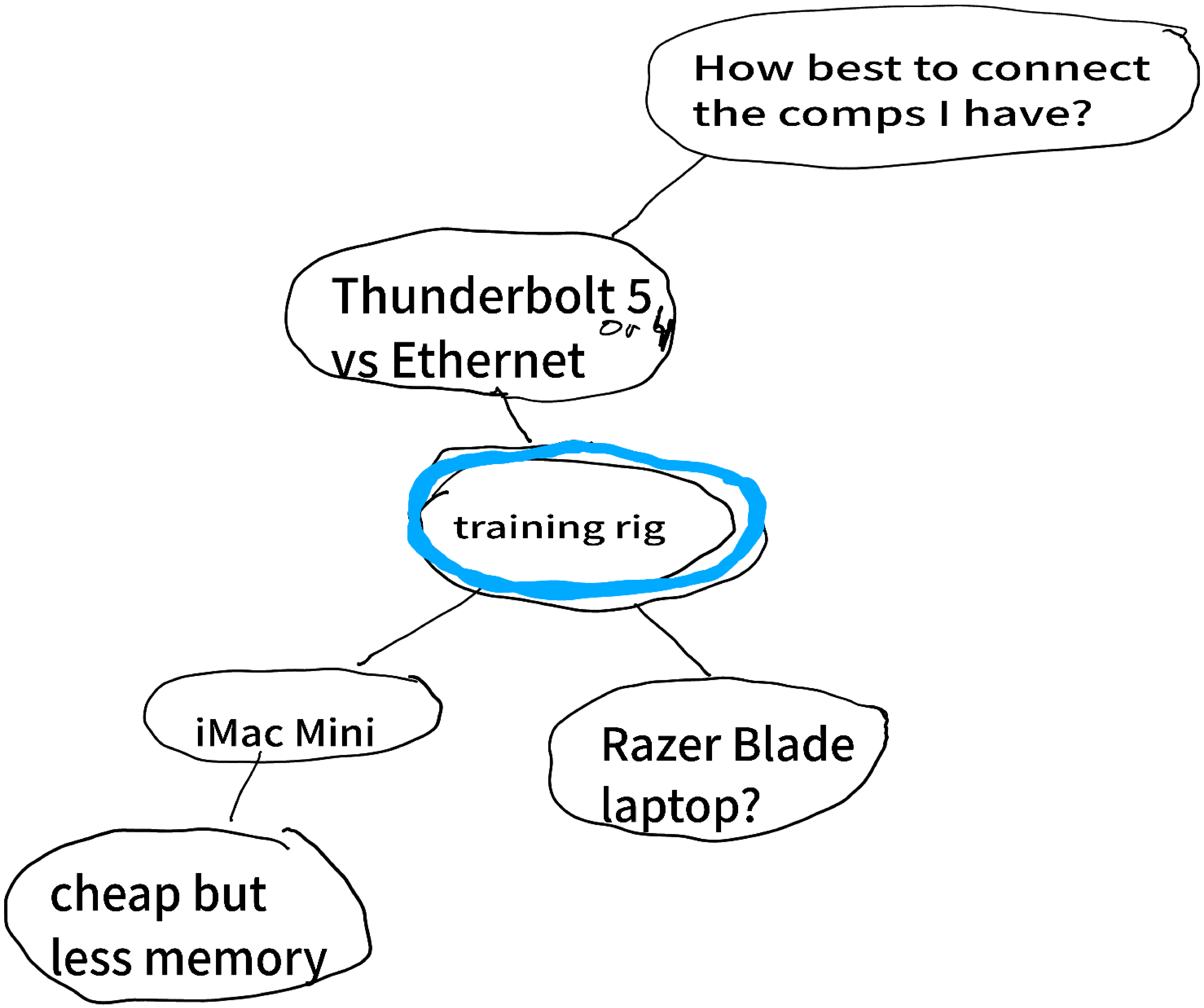
generic across backends

beam search and MCTS

program search

needs backend-specific fitness

2024.11.16



How best to connect the comps I have?

Thunderbolt 5 or 4 vs Ethernet

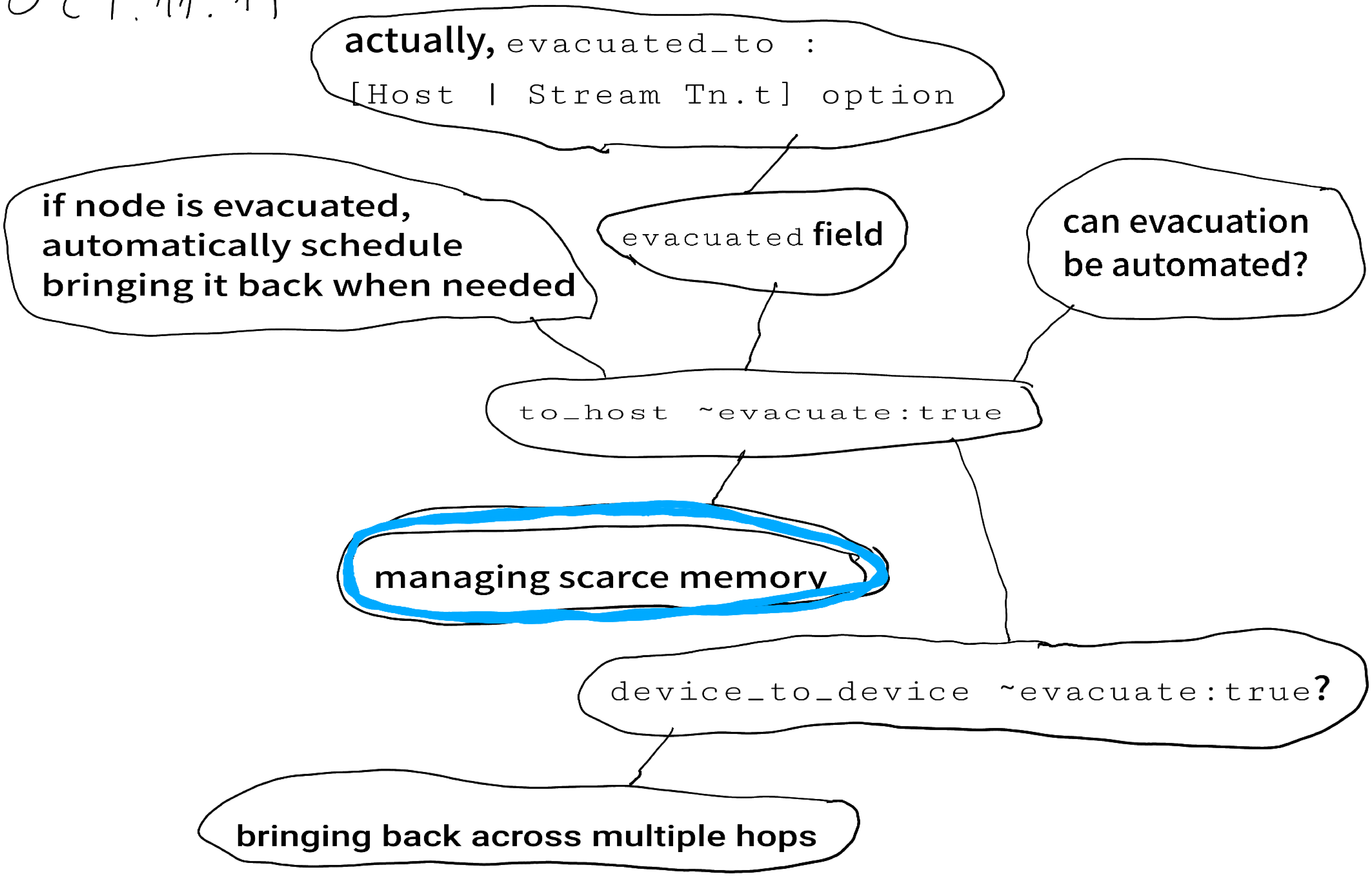
training rig

iMac Mini

cheap but less memory

Razer Blade laptop?

2024.11.11



actually, `evacuated_to : [Host | Stream Tn.t] option`

if node is evacuated, automatically schedule bringing it back when needed

`evacuated` field

can evacuation be automated?

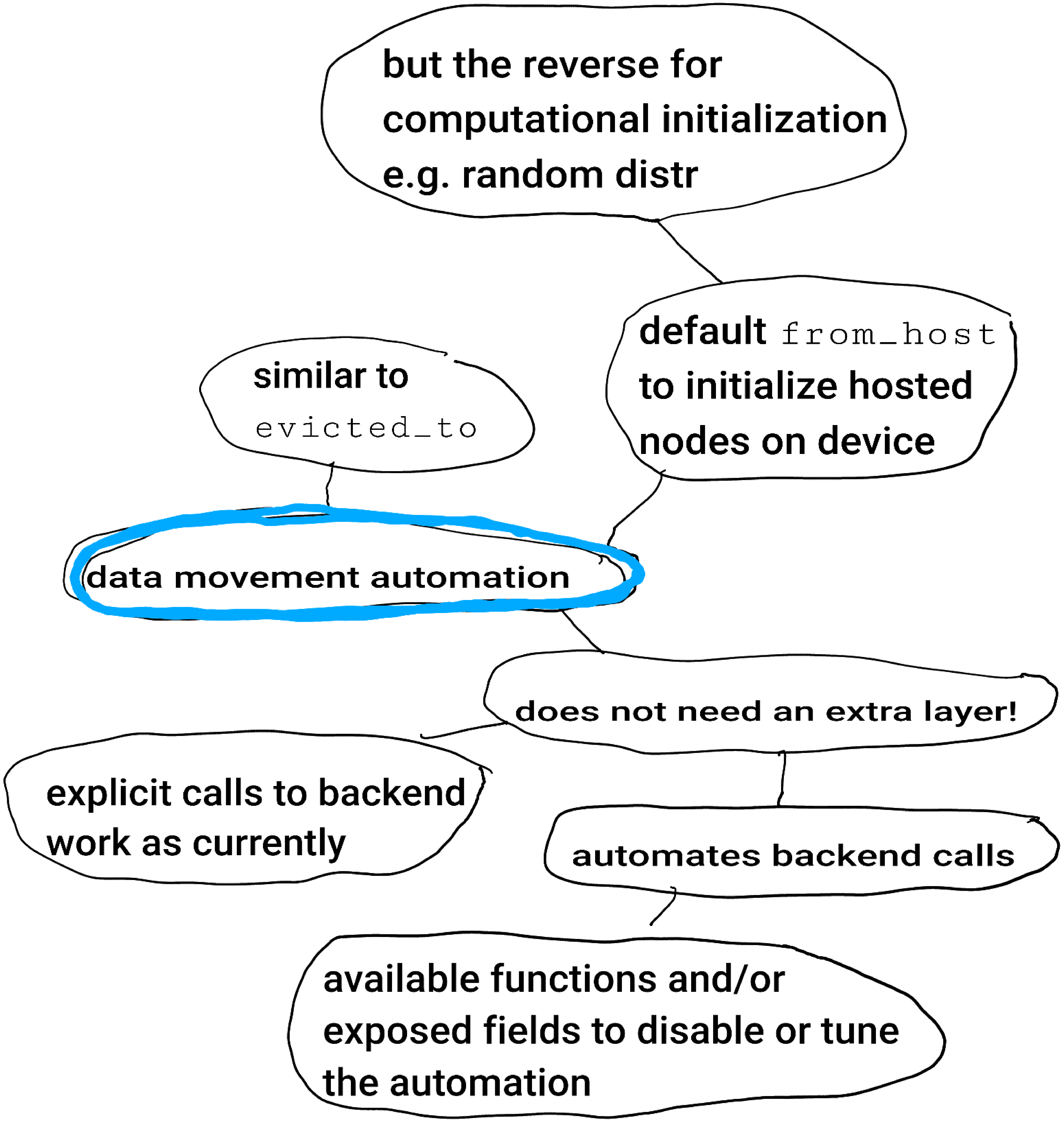
`to_host ~evacuate:true`

managing scarce memory

`device_to_device ~evacuate:true`?

bringing back across multiple hops

2024.11.12



but the reverse for computational initialization e.g. random distr

default `from_host` to initialize hosted nodes on device

similar to `evicted_to`

data movement automation

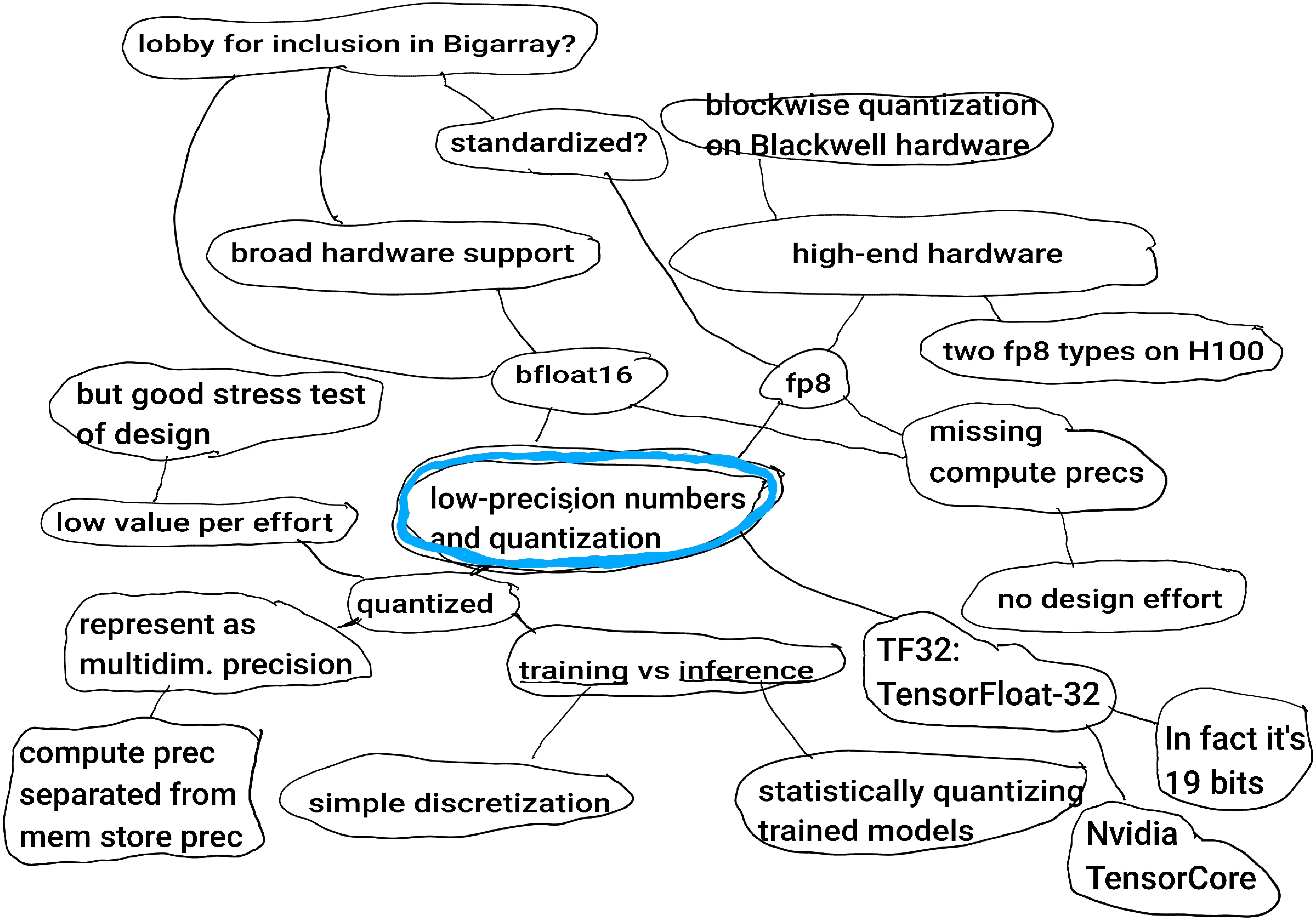
does not need an extra layer!

explicit calls to backend work as currently

automates backend calls

available functions and/or exposed fields to disable or tune the automation

2024.11.13



lobby for inclusion in Bigarray?

standardized?

blockwise quantization on Blackwell hardware

broad hardware support

high-end hardware

two fp8 types on H100

bfloat16

fp8

but good stress test of design

missing compute precs

low-precision numbers and quantization

low value per effort

no design effort

quantized

represent as multidim. precision

training vs inference

TF32: TensorFloat-32

In fact it's 19 bits

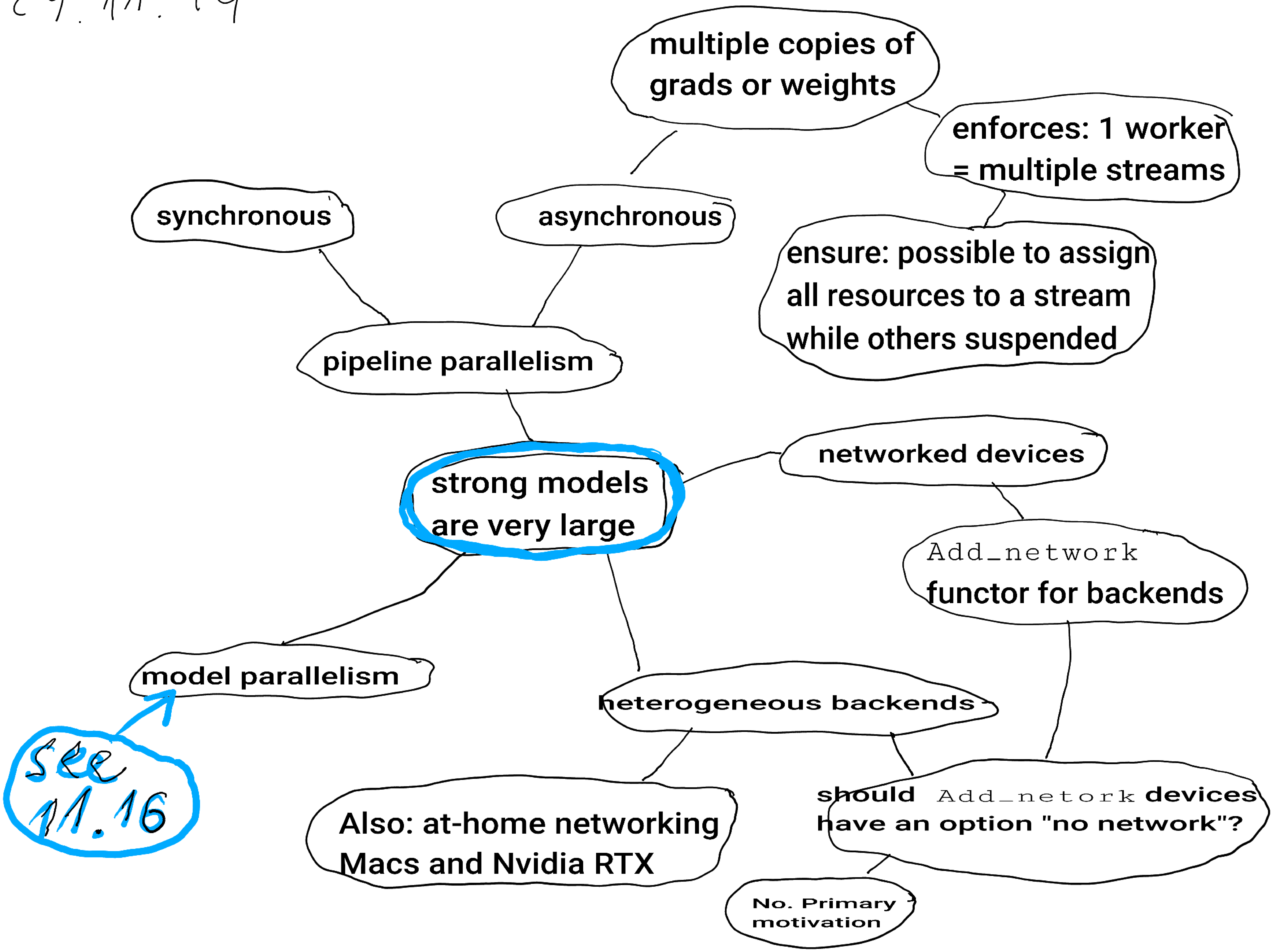
compute prec separated from mem store prec

simple discretization

statistically quantizing trained models

Nvidia TensorCore

2024.11.14



strong models are very large

pipeline parallelism

synchronous

asynchronous

multiple copies of grads or weights

enforces: 1 worker = multiple streams

ensure: possible to assign all resources to a stream while others suspended

networked devices

Add_network functor for backends

heterogeneous backends

Also: at-home networking Macs and Nvidia RTX

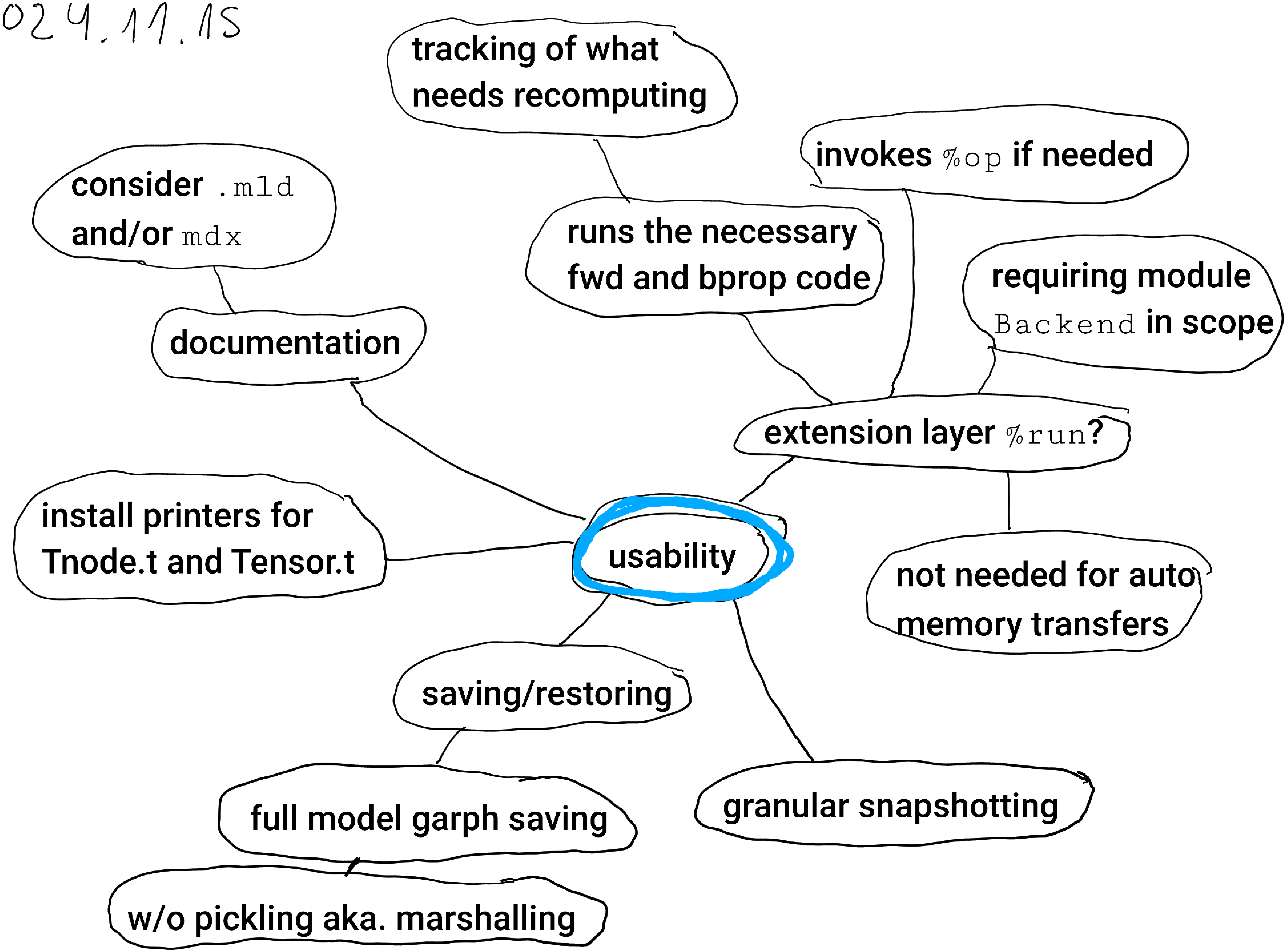
should Add_netork devices have an option "no network"?

No. Primary motivation

model parallelism

see 11.16

2024.11.15



usability

documentation

consider `.mld` and/or `mdx`

install printers for Tnode.t and Tensor.t

extension layer `%run`?

runs the necessary fwd and bprop code

tracking of what needs recomputing

invokes `%op` if needed

requiring module `Backend` in scope

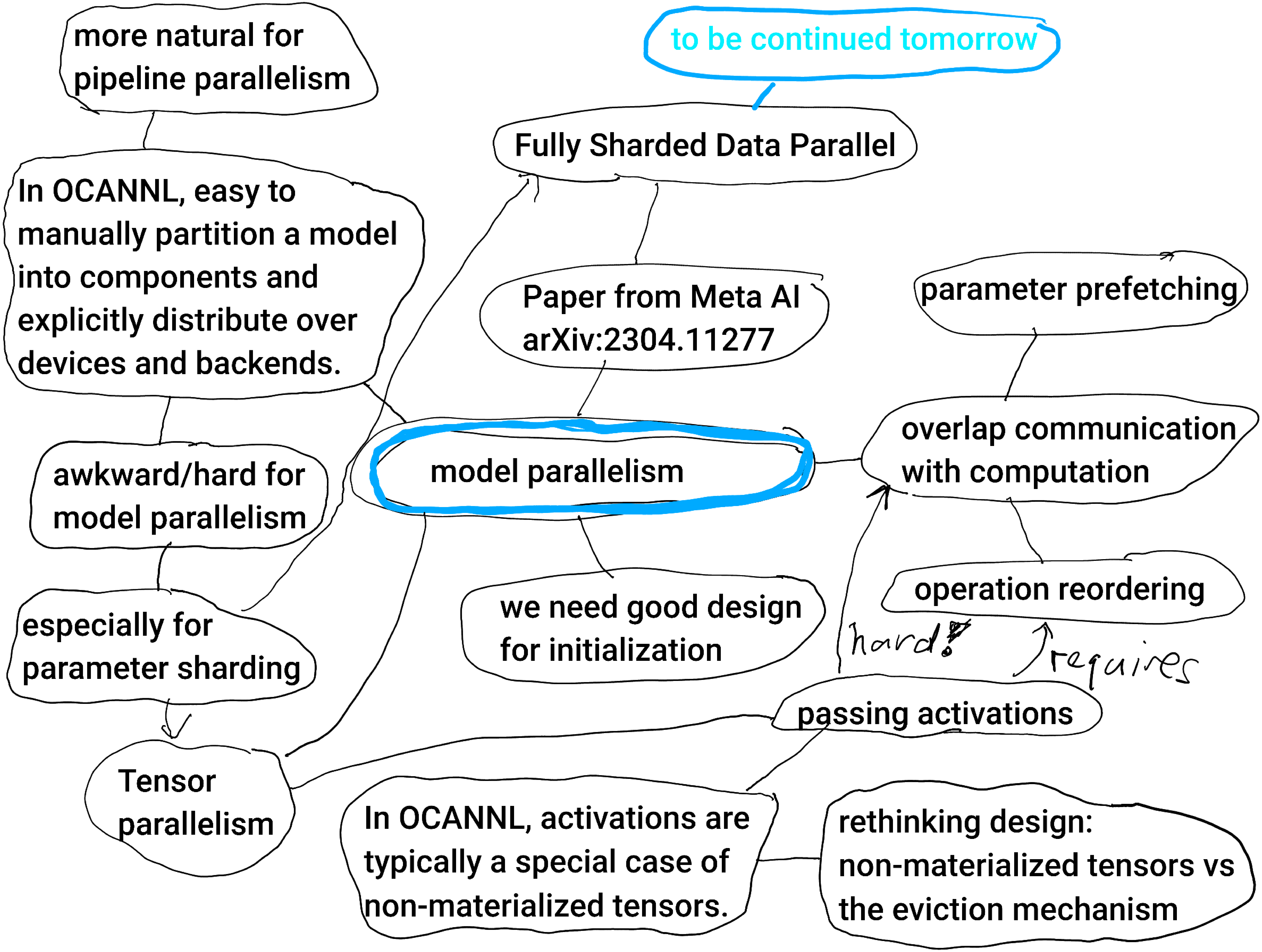
not needed for auto memory transfers

saving/restoring

full model garph saving

w/o pickling aka. marshalling

granular snapshotting



more natural for pipeline parallelism

to be continued tomorrow

Fully Sharded Data Parallel

In OCANNL, easy to manually partition a model into components and explicitly distribute over devices and backends.

Paper from Meta AI arXiv:2304.11277

parameter prefetching

awkward/hard for model parallelism

model parallelism

overlap communication with computation

especially for parameter sharding

we need good design for initialization

operation reordering

passing activations

Tensor parallelism

In OCANNL, activations are typically a special case of non-materialized tensors.

rethinking design: non-materialized tensors vs the eviction mechanism

2024.11.18

1. Each sub-module / model layer is a FSDP unit.
2. Each unit's non-shared parameters are flattened, concatenated and sharded across backends/devices.
3. Before the unit's computation, unshards required parameters. Afterward, deletes other shards' parameters.

communicates parameters & their gradients on demand, for unsharding & accumulation

crosses abstraction levels: bad fit for OCANNL design

great for balancing memory, computation, number of devices

Fully Sharded Data Parallel

parameter sharding

communicates activations: computation boundaries

downside: complicates model design with computational considerations

Tensor Parallelism

upside: keeps model design clean

tricky

automatically find good axes

worse fit for OCANNL design

manually: a slice operator indexed by backend/device/stream

hard work: find good axes to balance memory, computation, number of devices

fits with OCANNL design

2024.11.19, updated 2024.11.27

ideas for upcoming mind maps

convolutions and padding

1/3 done: tinygrad

design risk: interacts with the whole OCANNL, esp. shape inference

visually atractive examples

even LLMs: need padding

pipeline parallelism

mostly done

continues recent theme

program search in tinygrad

done

super important for design

Pallas: extension of JAX for writing custom kernels

done

mutable abstraction layer for JAX

what are blocks?

added 11.27

more deep dives

Karpathy's

llm.c

llama2.c

jackpeck's llama2.ml

LLM101n

nanoGPT

Keller Jordan's modded-nanogpt

keras

keras.core

2024.11.20

papers to read:
ZeRO arXiv:1910.02054
Zero Bubble arXiv:2401.10241
NanoFlow arXiv:2408.12757

more notes coming

pipeline with nano-batches at the granularity of operations

opimizer/gradient sharding

ZeRO: focus on minimizing per-GPU memory

ZeRO-DP is similar to FSDP, but lossy and much simpler

partitions optimizer states, discards gradient parts for other partitions

pipeline parallelism

PP most helpful for cross-server connections

asynchronous PP breaks optimizer semantics

Zero Bubble

optimizes microbatch scheduling of Forward, Backward, and Weight-gradient

layer-wise parallelizes the optimizer step by accumulating progressively

propagates global optimizer state of previous iteration while the next iteration is computing the initial forward steps

redoes the optimizer step if a global check fails for any layer (found INF, NaN, or gradient clip needed)

2024.11.21

program search in tinygrad

kernel transforming operations

operations usually have axis and amt (amount / part of)

LOCAL

converts part of a global axis to local

UPCAST

upcasted axis is computed or reduced at higher precision??

converts part of an axis into an innermost upcasted axis

UNROLL

converts part of or whole axis into an innermost upcasted axis

SWAP

swaps two axes in a tensor

TC

TensorCore's reduce axis is selected starting from first_reduce, the 2 inputs axes ending at first_reduce

applying TC applies PADTO, UNROLL for reduce axis, UPCAST, LOCAL for TensorCore's threads' axes

then optional "hand-coded" UPCAST and LOCAL

PADTO

adds padding to an axis

NOLOCALS

prohibits use of local axes

GROUPTOP

UPCASTMID

GROUP

related

fusing reduce ops??

in tinygrad, memory mode and projection semantics are assigned per-axis

blue - global dims

cyan - local dims

green - reduce-local

white - reduce-late upcasted

red - reduce loops

purple - reduce upcasted

yellow - normal upcasted

all of this is in codegen/kernel.py, beam_search itself is in engine/search.py

2024.11.22

why is all this in here? Where is the scheduling?

schedule.py: things to schedule, not scheduling.

scheduler in tinygrad

via graph rewriting

what is sizzle???

see later

prepares indexing (aka. movement ops), integrates indexing with computation

selects groups to fuse, vs. what to materialize

to_uop for non-const buffers is the ShapeTracker's view of the buffer

an UOp `is_scheduled` when the op is Ops.VIEW

ScheduleItem with disjoint input and output buffers

similar to OCANNL's inputs and outputs fields in `routine`

multiple outputs possible via `Ops.SINK` AST node, otherwise single output

upcoming design

PR 7065

Gets rid of LazyBuffer (replaced with UOp) and of engine/lazy.py

gets rid of indexing processing in schedule.py, instead exposes ShapeTracker methods in ops.py

tracks materialized buffers (i.e. `realized` in tinygrad) in ops.py

graph rewriting to push views below computations, collect buffers and kernels

in schedule.py

2024.11.23

Pallas: a JAX kernel language

imperative: explicit assignments

Refs were introduced to make JAX stateful even before Pallas, reused

on TPU

args are already SRAM (copied before running a kernel), then copied explicitly into registers before computation

on GPU

args are HBM/DRAM mem (global), explicitly copied into SRAM mem (local) before computation

more control over memory access

kernels parallelize over grids

vmap of pallas_call: adds extra grid axis

a BlockSpec projects an input or output to a block-slice view and threads/streams, for blockwise parallelism over grids

manually specified for each input & output

generalization of tiling

also needs explicit out_shape

dynamic slicing and masking

not implemented yet: alternatives to BlockSpec e.g. overlapping windows for convolutions

doesn't support:

conv_general etc. -- usually not on hardware

gather / scatter -- backends without noncontiguous memory reads / writes

GPU and TPU are not entirely interchangeable?

2024.11.24

Triton

`load`, `store` with dynamic slicing and mask

based on pointers

Pallas exposes them but on refs

both manual and auto-tune splitting into warps

autotune picks a config out of manually speced candidates

execution over a grid, as in CUDA and Pallas

each block must have 2^n elements, needs padding via mask

manual decoding of indices into per-block vectors of offsets for load/store, but otherwise functions over ndim arrays

tl.exp is __expf from CUDA (approximate)

nice idea: associate axes with strides to decouple tensor semantics from memory layout

sharing code across backends (CUDA, CDNA) but dedicated sections to compute e.g. num of threads per multi-processor

nice tutorials

maybe worth replicating triton.testing.(perf_report|Benchmark)

groups of blocks reduce memory transfers per same num of output blocks

not built-in

to be continued

2024.11.25

Triton part 2

counter-based Parallel Random Numbers

computes pseudo-random nums on the device with seed int32

manual

reducing across blocks in a group by locking at the end of a kernel

vs polyhedral DSLs

use affine *access functions*

support fusion, interchange, tiling, parallelization

loop transformations

large search space

not applicable to (structured-)sparse networks

vs scheduling DSLS

separate algo and schedule: tile splits, loop reordering and unrolling, parallel axes

TVM has built-in automatic scheduling

almost

auto optimizations

coalescing

orders threads within micro-tile to contiguous mem access

?

thread swizzling

manual!?

transforms row-major to column-major submatrix for each group-size rows

pre-fetching

async copy scheduling

shared mem synchronization

inserts barriers into GPU code by detecting read-after-writes and write-after-read

2024.11.26

tinygrad ShapeTracker

why many views per Tracker? -> tomorrow

vs teenygrad = 1/10th of tinygrad

translates all of NumPy syntax to ops

pure movement ops

RESHAPE

PERMUTE

PAD

fwd: NumPy pad
bprop: shrink

EXPAND

fwd: NumPy broadcast_to
bprop: reduce SUM

SHRINK

fwd: NumPy subarray view
bprop: pad

STRIDE

Decomposed using Flip for < 0 and a combination of Pad and Reshape for > 1

direct relevant ops are: EXPAND, CONTRACT, VIEW, REDUCE_AXIS

is a list of View objects

a View has: shape (i.e. dims), strides, offset, mask (begin-end per axis), whether it's contiguous

default strides assume shape is rightmost-major

canonicalized, e.g.: if mask uncovers just 1 index, convert to "no stride" and adjust the offset

assigned to op nodes except DEFINE_LOCAL/GLOBAL/VAR, BUFFER, CONST

VIEW has a ShapeTracker

other nodes inherit ShapeTracker from children; all children must have the same ShapeTracker!

VIEW doesn't have children, instead typically provides a view for the preceding DEFINE_GLOBAL

VIEW = non-copy movement op

2024.11.27

tinygrad followup

multigpu via sharding

MultiLazyBuffer with per-device LazyBuffers

ends up duplicating the part of the compute graph below reducing of the sharded axes

stores per-device bounds of the sharded axis

CAST and BITCAST

dedicated float4 support

BITCAST frist takes address and pointer-casts the address

UPCAST UOp actually means UNROLL!

What is UPCAST Opt (i.e. in program search)?

STORE: retain after kernel finishes

its ScheduleItem forms a kernel

DEFINE_GLOBAL args: position in parameter list, param name, mutability

Variable

axis dimensions placeholder

can create_schedule_with_vars perform some shape inference?

can remain symbolic

generates symbolic mask guard

passed as a kernel param

has range

this range is used to render loops in Linearizer