TDT 4200 Fall 2024
Parallel Computing
Lect 9: MPI_Scatter/Gather – time analysis/Performance, More on Caching

Prof. Anne C. Elster, PhD

Dept. of Computer Science (IDI)

Norwegian Univ. of Science & Technology

(NTNU Trondheim, Norway)

&

Univ. of Texas at Austin (Senior Visiting Scientist)

TDT4200 F2024 Course Information

– See also https://www.ntnu.edu/studies/courses/TDT4200/

- Instructor: Professor **Anne C. Elster** (elster@ntnu.no)
- Recitation lectures and assignments: **Tobias Dyngeland**
- Course supporter: Assoc. Prof. **Jan Christian Meyer**

- Web page: http://www.idi.ntnu.no/~elster/tdt4200/fall2024 TBA

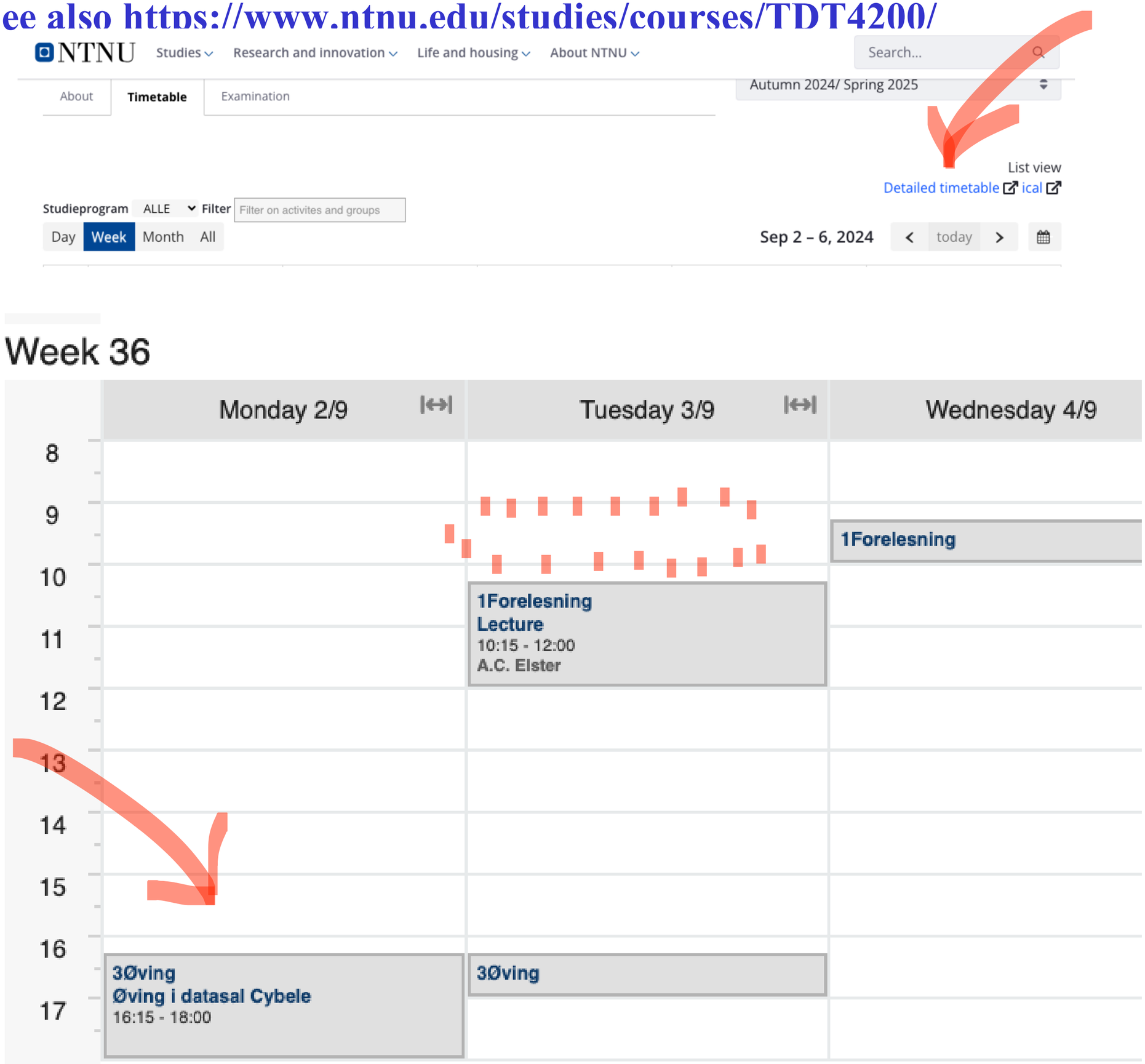
- **Lectures:**
  - Tuesdays: 10:15-12:00 in R5
  - Wednesdays: 9:15-10:00 in R7

- **Recitation lectures (Øvingstime):**
  - Tuesdays: 16:15-17:00 in Kjel 5
  - ~~Wednesdays: 8:15-9:00 in Kjel 1~~ – **replaced by Cybele hrs on Mondays:** (Updated under "Detailed TimeTable")

- **Lab help Mondays 16-18 @ Cybele**
- **Course Tool: BlackBoard**

TDT4200 F2024 Course Information

– See also https://www.ntnu.edu/studies/courses/TDT4200/



Course Information – continued

See also https://www.ntnu.edu/studies/courses/TDT4200/

NOTE: Compulsory assignments:

– **You need to do and pass ALL Problem Sets/Exercises in order to take the final**

… also those that are Pass/Fail!!

.

BlackBoard – Problem Sets -- Updated dates!

Problem Sets -- Tentative Dates

Tentative schedule for the problem sets :

Problem set (Exercise)	**Available (Tuesdays in Recitation)**	**Due Mondays 10pm**	**Topic**	**Grading**
PS 0	Aug 20	Sep 2/9*	C - intro -- pointers ++ (optional, but highly recommended)	Pass/Fail -- Optional
PS 1a	Aug 27	Mon Sep 9*	C - Wave Equation	Pass/Fail -- Required
PS 1b	Aug 29	Mon Sep 9*	MPI Intro (Optional, but highly recommended)	Pass/Fail -- Optional
PS2	Sep 10	Mon Sep 16	MPI 1D Wave eqn	Pass/Fail - Required
PS3	Sep 17	TBD	**MPI 2D Wave eqn**	**Graded - Required**
PS4	Oct 1	TBD	Pthreads/OpenMP	Pass/Fail - Required
PS5	TBD	TBD	CUDA Intro	Pass/Fail - Required
PS6	TBD	TBD	**CUDA 2D Wave eqn**	**Graded - Required**

* You can technically submit these until the end of the add/drop period, but are STRONGLY advised to do them ASAP.

Textbook

BlackBoard – Forum

Thread Actions | Collect | Delete

	DATE	THREAD	AUTHOR	STATUS	UNREAD POSTS	UNREAD REPLIES TO ME	TOTAL POSTS
☐	9/1/24 6:54 PM	Problem with movie creation	**Arthus Guy Daimez**	Published	0	0	3
☐	8/28/24 1:58 PM	Issue Running Assignment 1 on Snotra Server	**Simone Deidier**	Published	0	0	2
☐	8/25/24 8:13 PM	Can i deliver assignment 0 on the 2nd of september?	**Kristian Sørli**	Published	0	0	4
☐	8/23/24 2:30 PM	Problem Set resubmission	**Eivind Kløvjan**	Published	0	0	2
☐	8/21/24 10:57 AM	Comments/Questions -- Main lectures	**Anne Cathrine Elster**	Published	0	0	2
☐	8/20/24 4:40 PM	Recommendations for learning C	**Henriette Marie Eltvik**	Published	0	0	3
☐	8/20/24 1:00 PM	Exercise submission	**Guillaume Carraux**	Published	0	0	3
☐	8/19/24 1:53 PM	Welcome to the TDT4200 Forum / Discussion Board for Fall 2024	**Anne Cathrine Elster**	Published	0	0	1

Thread Actions | Collect | Delete

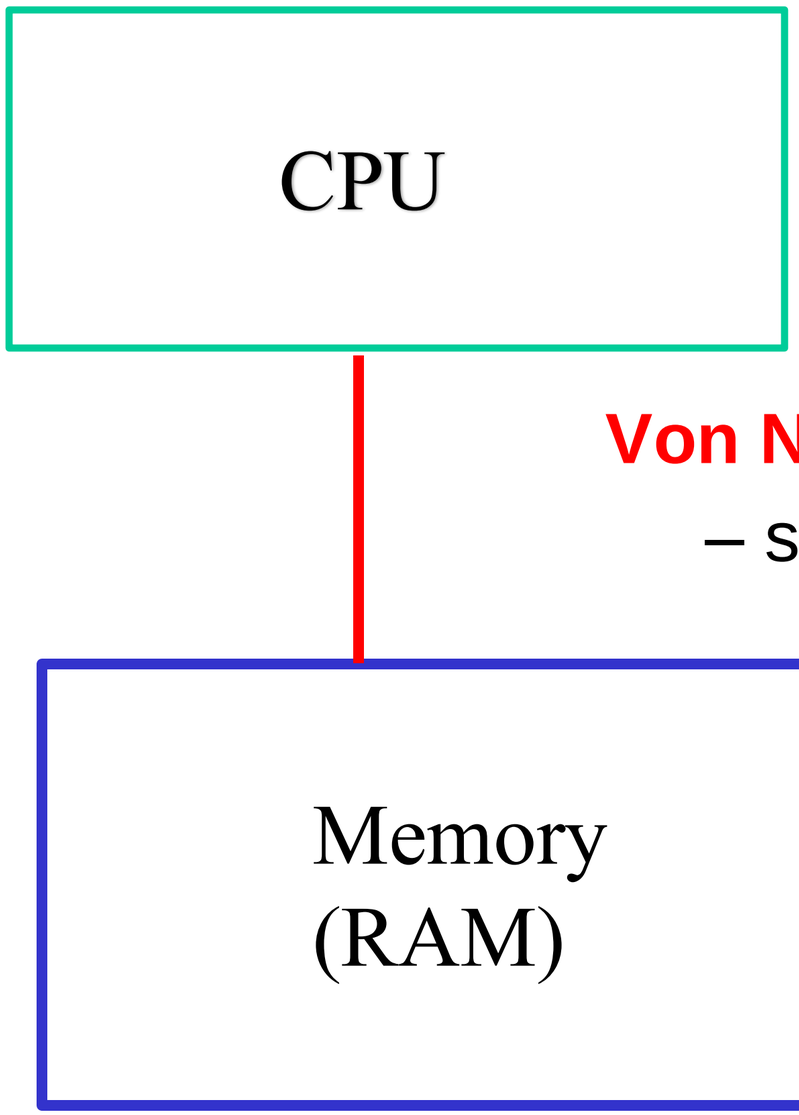
Outline

- Course Info.

- **MPI Scatter/gather and timing –** on blackboard and JCM slides13, slides 14-32
  - Timing, incl. Latency & Bandwidth
  - Implementation impact on performance (serial vs trees etc)

- **More on Caching –** on black board, these slides + JCM slides24

The Von Neuman Architecture
→ sequential computer (SISD)



Von Neuman bottleneck

– since CPU 500-1000 times faster than RAM!

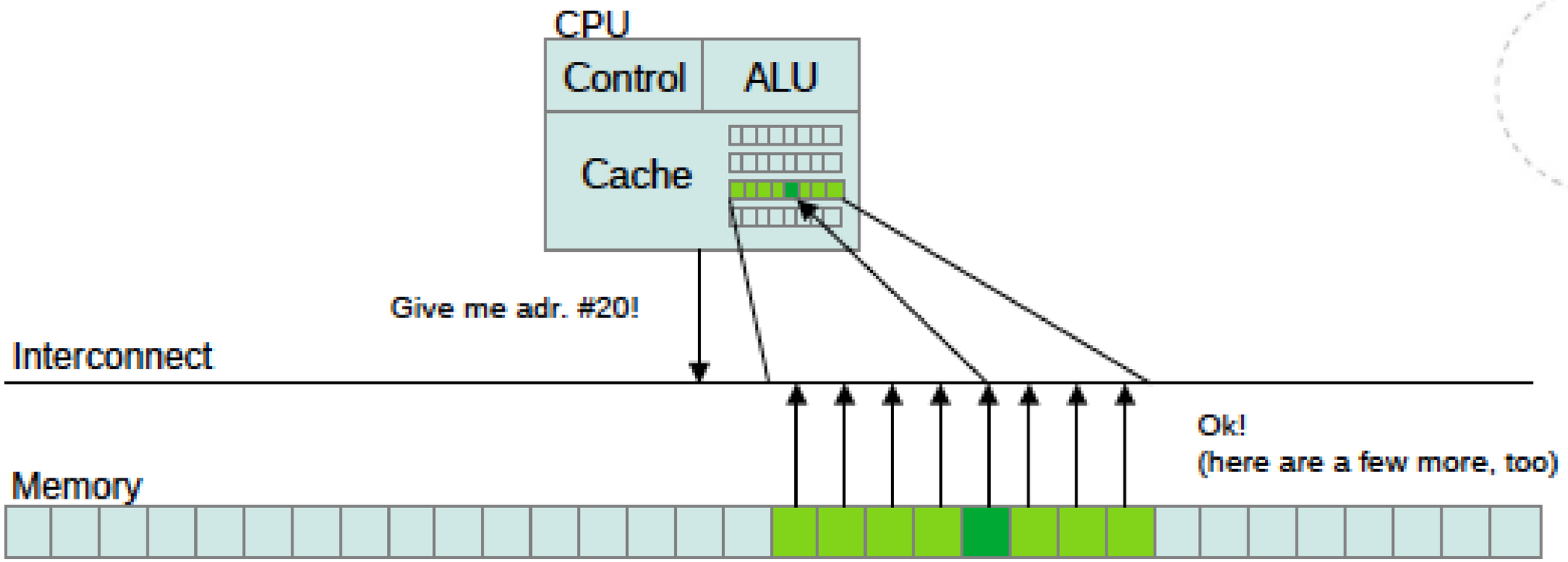
→ Add caches!

(Illustrated system with 3 cache levels on the blackboad)

Caching

(from JCM slides26)

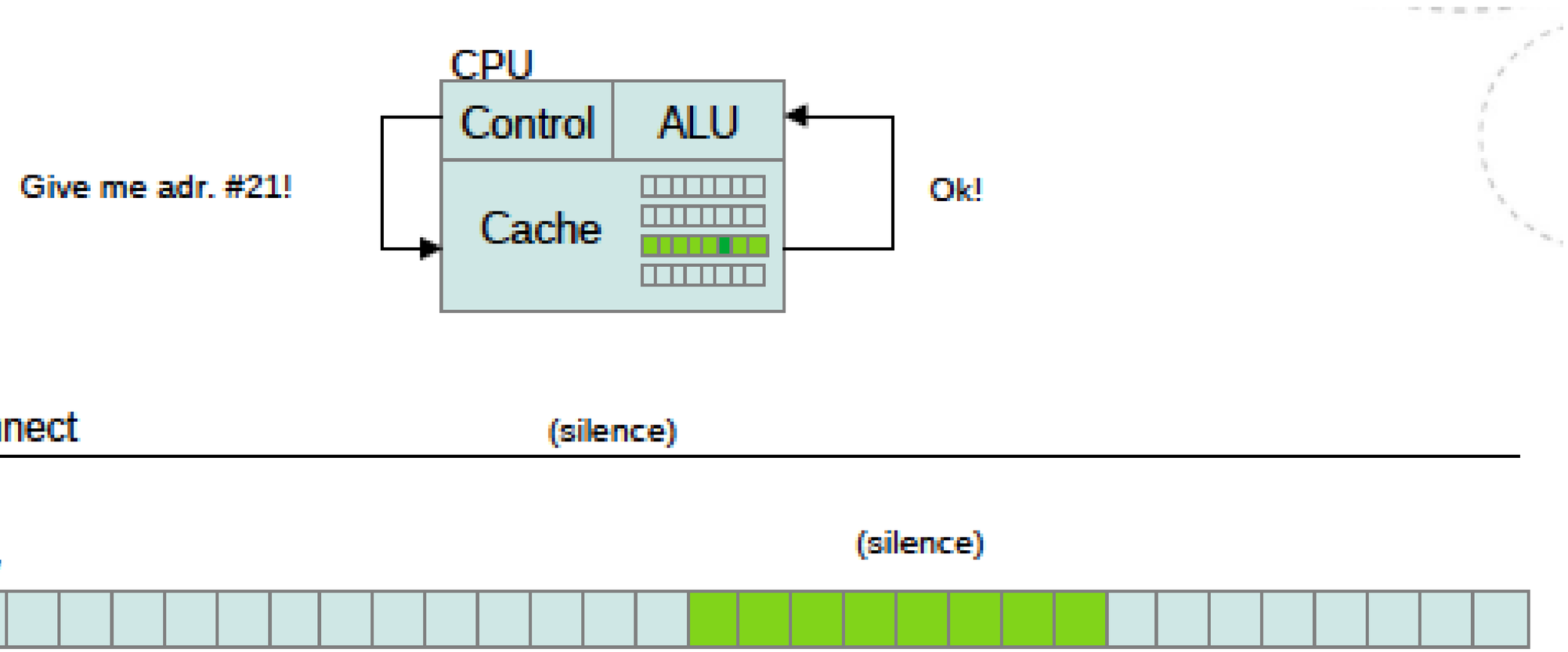
Reading un-cached value:



Caching

(based on JCM slides26)

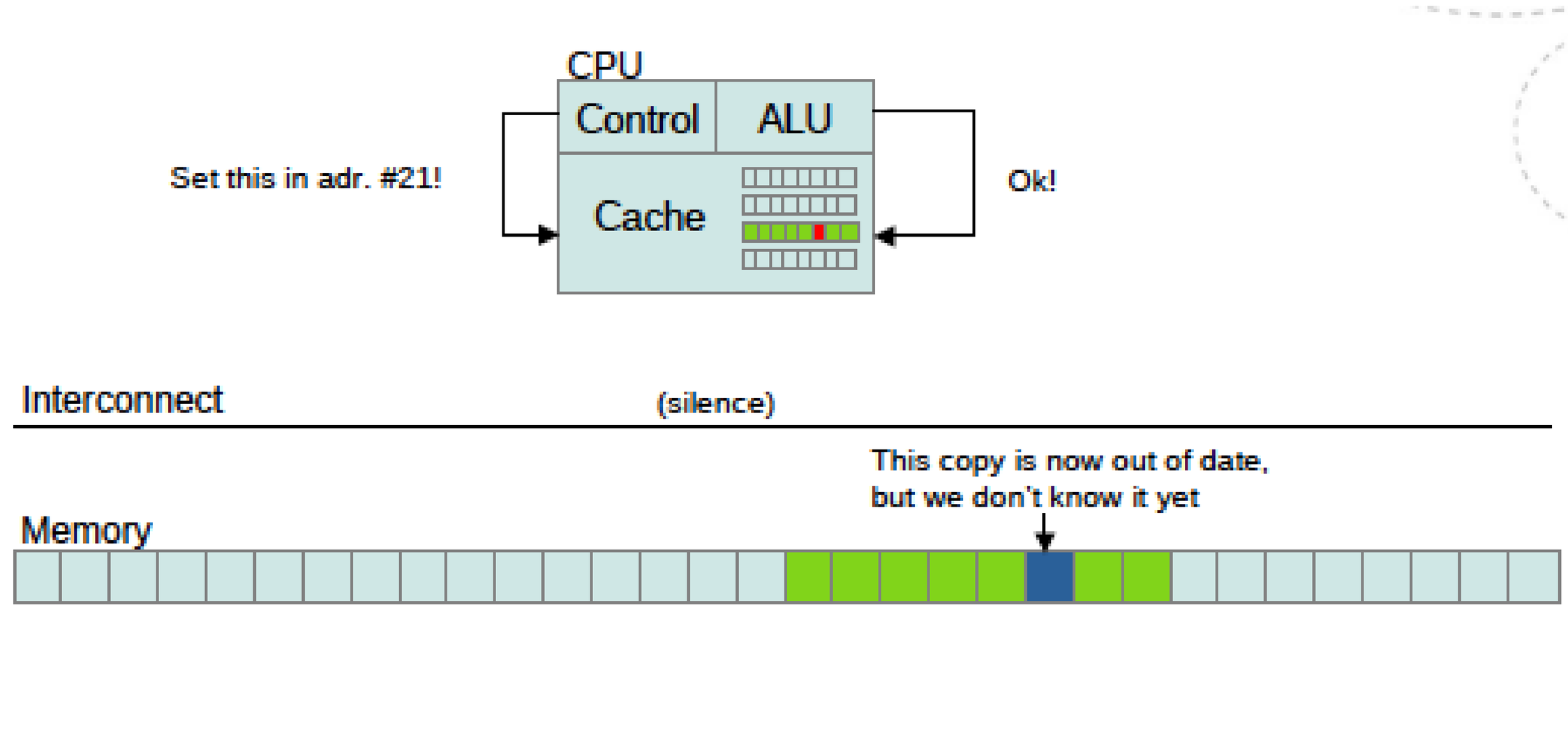
Reading another (now cached) value:



Caching

(based on JCM slides26)

Writing a cached value:



Caching – writing cached value

(based on JCM slides26)

We have 2 options:

1) Push write operations straight through cache memory, and update main memory ASAP

+ Simple and inexpensive implementation

- May create constant memory traffic

2) Delay write operations in memory, continue working with the cached copy

+ Doesn’t do unneccessary work

- Requires more complex circuitry and wiring

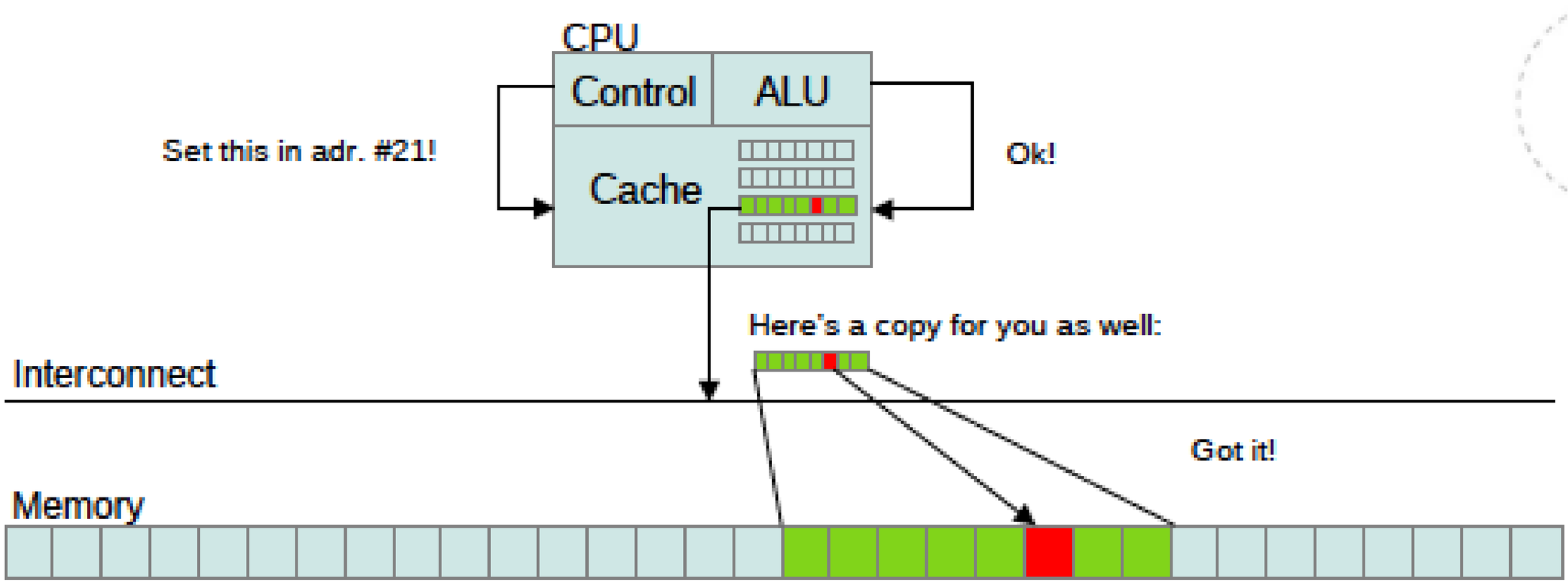
(Similar tradeoff to eager vs. lazy evaluation in programming languages)

Caching – writing cached value

(JCM slides26 – slide 8)

Write-through caching

- Write-through caches immediately pass their updates to main memory via the interconnect



Caching – writing cached value

(based on JCM slides26)

See JCM slides24, slides 8-24

MPI is SPMD!

- Single Program, Multiple Data
- Model proposed in 1984
  - by **Dr. Frederica Darema** (IBM)
    - later @ DARPA, NSF and now Director of US AirForce Research.
    - also known for Dynamic Data Driven Application Systems (DDDAS) proposed in 2000.

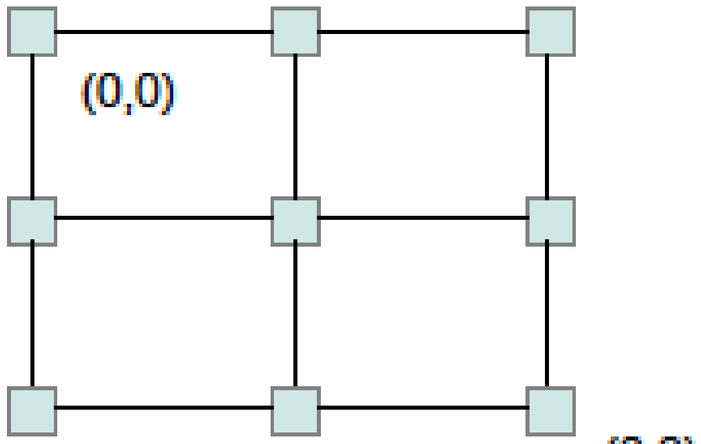
https://www.af.mil/About-Us/Biographies/Display/Article/3271084/dr-frederica-darema/

- Can be thought of as subcategory of MIMD
- **MPI is SPMD!**

Unlike SIMD, can use control statements like if/else to do execute different instructions concurrently

MPI Virtual Topologies (from JCM Slides 08)

- The world communicator has no internal structure, everyone just gets a number for rank
- MPI lets you declare communicators that have structure, *e.g.* the *Cartesian* flavor, where every rank has a set of coordinates:



- This way you can send/receive messages with “rank at (1,1)” instead of having to calculate an indexing scheme yourself

- We can get communicators shaped like arbitrary graphs as well, but this rectangular thing is common

MPI Communication modes

(based on JCM Slides 08)

Point-to-point messages (e.g. MPI_ Send and MPI_Recv) can be sent with **4 different guarantees for how they are transmitted**:

- **Standard** (implementation default, more on this later)
- **Synchronized** (Send-function will not return until reception is acknowledged)
- **Buffered** (Explicitly manage the memory that's used for sending/receiving)
- **Ready** (Assume that the receiver has already initiated the receive)

MPI Non-Blocking Communication

(based on JCM Slides 08)

- Usually, send and receive operations cause the program to stop and wait for the message to come through, and only resume the program afterwards

- This is not 100% true, but close enough for now

- Non-blocking sending and receiving immediately returns a request instead, so that you can continue calculating

- In order to make sure that the message has gone/come through, you **must issue a wait-for-completion** call for the request later on
  - Whenever you can no longer proceed without the comms being complete

More MPI

MPI Collectives :

- MPI_Barrier,
- MPI_Bcast,
- MPI_Scatter /MPI_Gather (+ all-gather /all-scatter)

MPI Derived datatypes:

- combing several datatypes in a reusable buffer

MPI Parallel I/O

- Lecture with Jan Chr. Meyer on Sept 11!

MPI functions -- OpenMPI

MPI API
(section 3 man pages)

MPI
MPIX_Allgather_init
MPIX_Allgatherv_init
MPIX_Allreduce_init
MPIX_Alltoall_init
MPIX_Alltoallv_init
MPIX_Alltoallw_init
MPIX_Barrier_init
MPIX_Bcast_init
MPIX_Exscan_init
MPIX_Gather_init
MPIX_Gatherv_init
MPIX_Neighbor_allgather_init
MPIX_Neighbor_allgatherv_init
MPIX_Neighbor_alltoall_init
MPIX_Neighbor_alltoallv_init
MPIX_Neighbor_alltoallw_init
MPIX_Query_cuda_support
MPIX_Reduce_init
MPIX_Reduce_scatter_block_init
MPIX_Reduce_scatter_init
MPIX_Scan_init
MPIX_Scatter_init
MPIX_Scatterv_init
MPI_Abort
MPI_Accumulate
MPI_Add_error_class
MPI_Add_error_code
MPI_Add_error_string
MPI_Address
MPI_File_call_errhandler
MPI_File_close
MPI_File_create_errhandler
MPI_File_delete
MPI_File_f2c
MPI_File_get_amode
MPI_File_get_atomicity
MPI_File_get_byte_offset
MPI_File_get_errhandler
MPI_File_get_group
MPI_File_get_info
MPI_File_get_position
MPI_File_get_position_shared
MPI_File_get_size
MPI_File_get_type_extent
MPI_File_get_view
MPI_File_iread
MPI_File_iread_all
MPI_File_iread_at
MPI_File_iread_at_all
MPI_File_iread_shared
MPI_File_iwrite
MPI_File_iwrite_all
MPI_File_iwrite_at
MPI_File_iwrite_at_all
MPI_File_iwrite_shared
MPI_File_open
MPI_File_preallocate
MPI_File_read
MPI_File_read_all
MPI_Ineighbor_allgather
MPI_Ineighbor_allgatherv
MPI_Ineighbor_alltoall
MPI_Ineighbor_alltoallv
MPI_Ineighbor_alltoallw
MPI_Info_c2f
MPI_Info_create
MPI_Info_delete
MPI_Info_dup
MPI_Info_env
MPI_Info_f2c
MPI_Info_free
MPI_Info_get
MPI_Info_get_nkeys
MPI_Info_get_nthkey
MPI_Info_get_valuelen
MPI_Info_set
MPI_Init
MPI_Init_thread
MPI_Initialized
MPI_Intercomm_create
MPI_Intercomm_merge
MPI_Iprobe
MPI_Irecv
MPI_Ireduce
MPI_Ireduce_scatter
MPI_Ireduce_scatter_block
MPI_Irsend
MPI_Is_thread_main
MPI_Iscan
MPI_T_init_thread
MPI_T_pvar_get_info
MPI_T_pvar_get_num
MPI_T_pvar_handle_alloc
MPI_T_pvar_handle_free
MPI_T_pvar_read
MPI_T_pvar_readreset
MPI_T_pvar_reset
MPI_T_pvar_session_create
MPI_T_pvar_session_free
MPI_T_pvar_start
MPI_T_pvar_stop
MPI_T_pvar_write
MPI_Test
MPI_Test_cancelled
MPI_Testall
MPI_Testany
MPI_Testsome
MPI_Topo_test
MPI_Type_c2f
MPI_Type_commit
MPI_Type_contiguous
MPI_Type_create_darray
MPI_Type_create_f90_complex
MPI_Type_create_f90_integer
MPI_Type_create_f90_real
MPI_Type_create_hindexed
MPI_Type_create_hindexed_block
MPI_Type_create_hvector
MPI_Type_create_indexed_block

Observing re. MPI

(from JCM slides09)

Every MPI function is called something like

MPI_Abcd_efg_h

- "MPI_" to begin with
- First letter in the function name is capitalized
- The rest of the name is all in lowercase, with underscore separation

- MPI uses arguments to pass variables in and out of functions
  - For the vast, vast majority of functions, return value is an error code that indicates whether the function completed in style or not
  - In order to obtain the answer from a function, you pass it a pointer to an area you have sized up to contain it, and let the function write it there

Observing re. MPI

(from JCM slides09)

Why use pointer-arguments instead of C's own return values?

- There is actually a reasonable rationale behind this, you will find that system libraries and many other libraries do it as well
- The purpose is to give the programmer complete control over allocation
- If you're coming from an OO language, it's tempting to build 'constructors' for your structs like this:

```
my_thing * create_thing( int a, int b, int c) { /* malloc in here */ }
void destroy_thing ( my_thing *dead ) { free ( dead ); }
```

  and use them like this

```
my_thing *newThing = create_thing (1,2,3);
destroy_thing ( newThing );
```

- This will force all my_things into the heap

Parallel Computing is Fun!