Sketch of a Distributed Education Environment Based on Mathematica

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Introduction

Distributed Education

Conventional view: *video conferencing*.

- Audience sited in various remote locations.
- Presentation is recorded and multicast.
  - Audio and video.
  - Special lines (ISDN, ATM) or Internet (MBone).
- Transmitted information:
  - Lecturer's speech and gestures.
  - Images of blackboard drawings/slides.
- Backchannels
  - Feedback from audience.
  - Group discussions.

What you see or hear is (all) what you get.

Computer-Supported Mathematics

Mathematica© (*Mma*) as a tool for mathematical education.

- Sessions based on *Mma notebooks*.
  - Multimedia documents with executable contents.
  - Composed of notebook *cells*.
- Frontal presentation: teacher equipped with computer.
  - Projects notebook to screen.
  - Runs examples in notebooks.
- Interactive lessions: students equipped with computers.
  - Use given notebooks as course material.
  - Run predefined examples in notebooks.
  - Program and run own examples in notebooks,
  - Return notebooks as assignment results.

How to integrate into distributed sessions?

Distributed Computer-Supported Education

Surface view: *shared screen*.

- Lecturer's user interface is monitored.
- Screen drawing operations are multicast.
- All participants receive identical screen copies.
- Input control may be switched among participants.
- *Shared X*, ...

Screen images as a supplement to video.

Distributed Computer-Supported Education

Beyond the surface: *shared tools*.

- Education software becomes multi-user capable.
- Distributed presentation.
- Distributed interaction.
- Distributed collaboration.

Collaborate by the underlying objects.

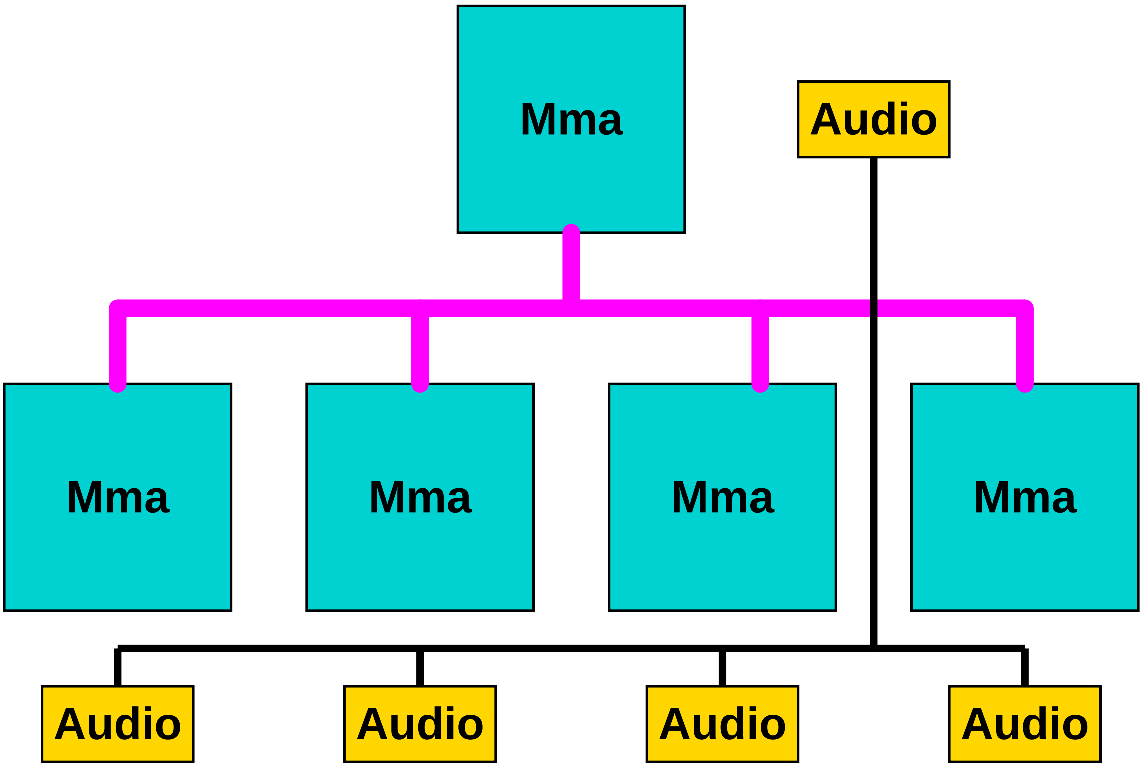
Distributed Computer-Supported Mathematics

Vision: *Mma* as a tool for distributed mathematics.

- All participants run copies of *Mma*.
- Lecturer presents notebooks and runs examples.
  - Contents forwarded to participants.
  - Activities forwarded and executed locally.
- Participants exercise locally.
  - Independent work on local notebook copies.
  - Results stored in local copies.
- Participants present results.
  - Temporary role of lecturer.
- Lecturer collects results.
  - Retrieval of generated notebooks.

How to implement/coordinate such distributed sessions?

Distributed Mathematics



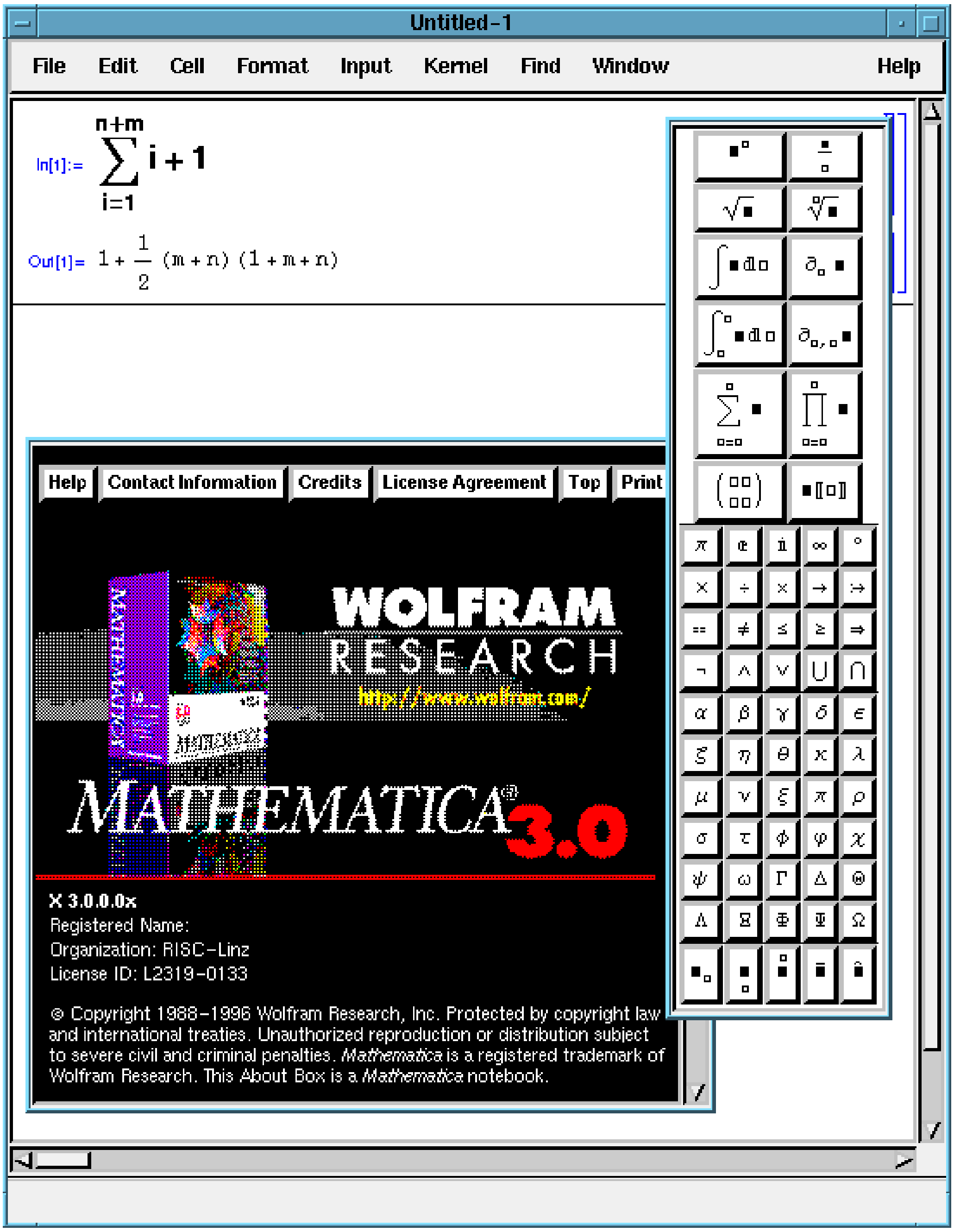
Audio/video support is a separate issue.

Scratching the Surface

Structure of Mathematica

- Frontend process.
  - Interaction with user.
  - Notebook manipulation.
- Kernel process.
  - Actual computation.
  - Simple text-base interface.
- Communication by *MathLink*.
  - Message passing model.
  - C-based programming library.
  - Implemented by sockets.

Notebook Frontend

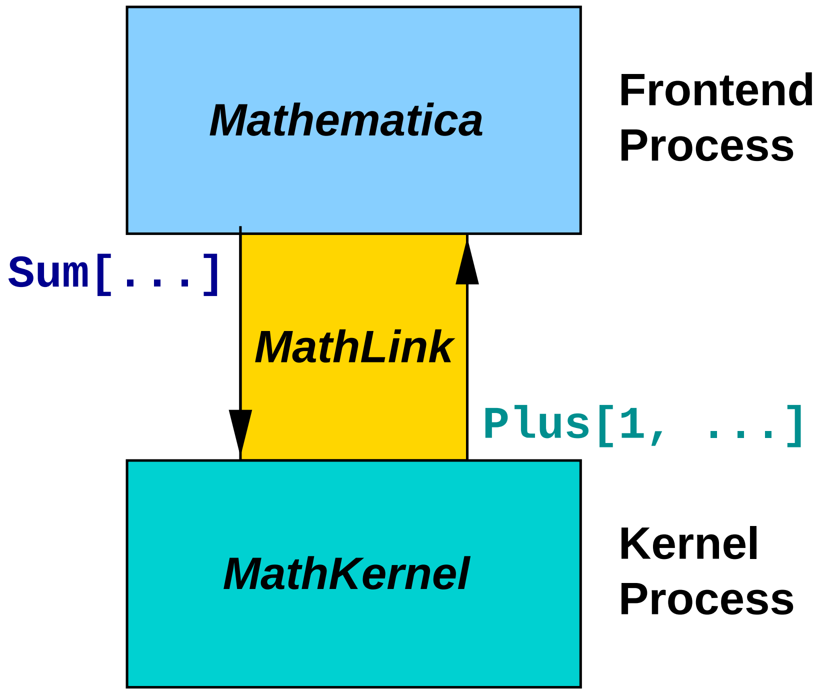


Kernel

```
iris!21> math
Mathematica 3.0 for Silicon Graphics
Copyright 1988-96 Wolfram Research, Inc.
-- Motif graphics initialized --

In[1]:= Sum[i+1, i, 0, n+m]

                     (m + n) (1 + m + n)
Out[1]= 1 + m + n + -------------------
                            2

In[2]:=
```

MathLink

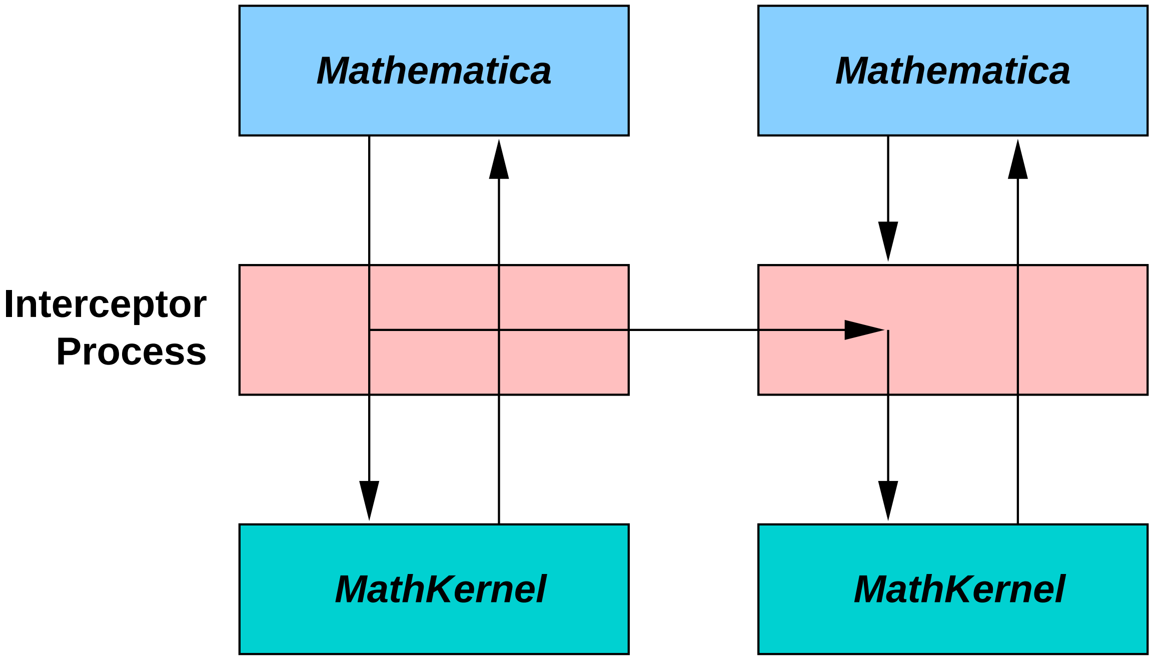
```
iris!24> ps -ef | fgrep math
schreine 29681 /usr/.../Mathematica -topDirect
schreine 29685 /usr/.../MathKernel -mathlink
```



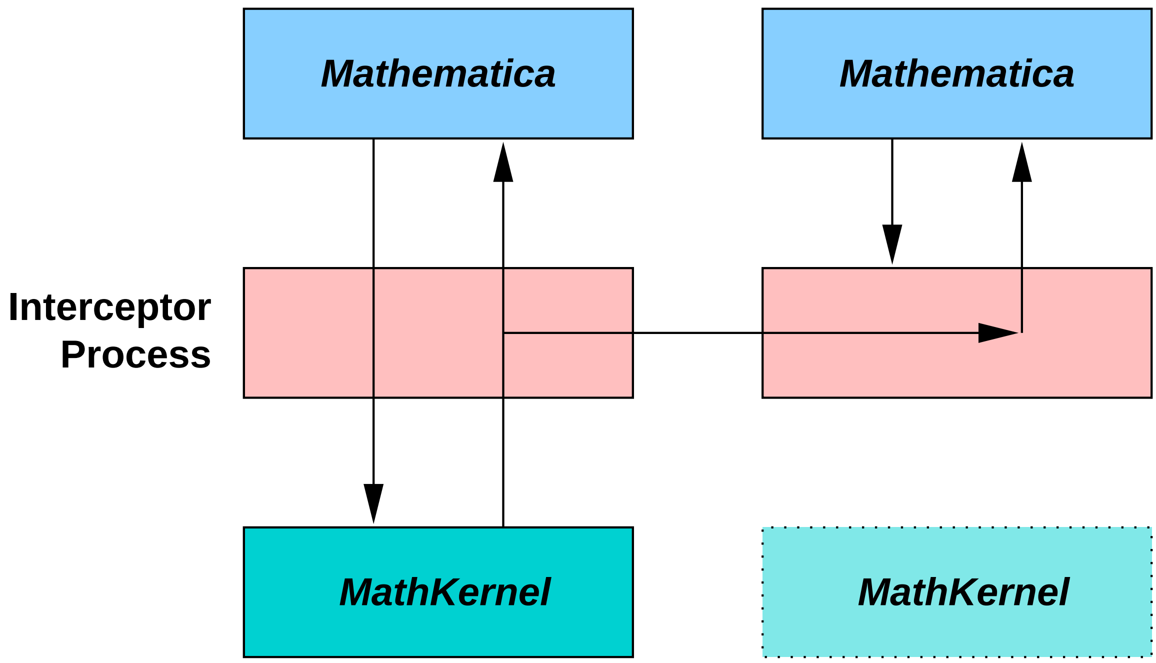
Basic Idea

- Intercept communication between frontend and kernel.
  - Transmit to other instances of *Mma*.
- Forward interception
  - Messages from frontend forwarded to other kernels.
  - Kernels return identical results to their frontends.
- Backward interception
  - Messages from kernel forwarded to other frontends.
  - Frontends receive result from single kernel.

Forward Interception



Backward Interception



Forward Interception

First analysis:

\+ *Independent* execution possible.

- Multicasting puts kernels into same state.
- Independent execution may then be started.

– Problems of *consistency*.

- Reliable multicasting required.
- Crashes may occur.
- Latecomers must be considered.
- Independent execution yields different states.

Backward Interception

First analysis:

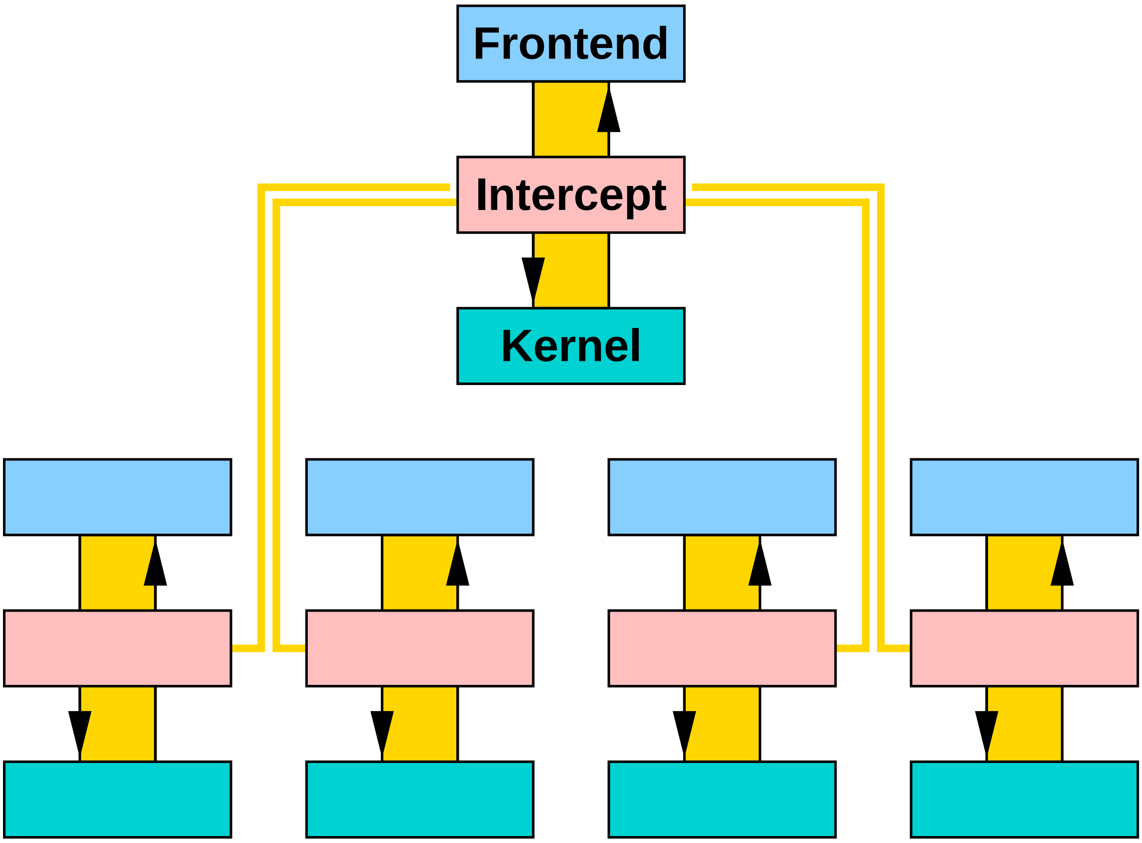
+ Single kernel required.

  - Participants only need frontends.
  - No consistency problems among kernels.

–

  - Output potentially huge.
  - No independent execution possible.
  - Also frontend has state and consistency problems!

Design Sketch



Going Inside

Notebooks in Frontend

A couple of *bad* news:

- Window events handled by *frontend*:
  - Scrolling within a notebook.
- Basic notebook manipulation takes place in *frontend*:
  - Reading a notebook from file.
  - Opening/closing a notebook cell.
  - Selecting and updating cell contents.
  - Navigating in a notebook.
- Only cells to be evaluated are transferred to *kernel*:
  - `Shift-Return` key.
  - `EvaluatePacket` sent to kernel.
  - `ReturnPacket` returned to frontend.

Notebooks in Kernel

Now for the *good* news:

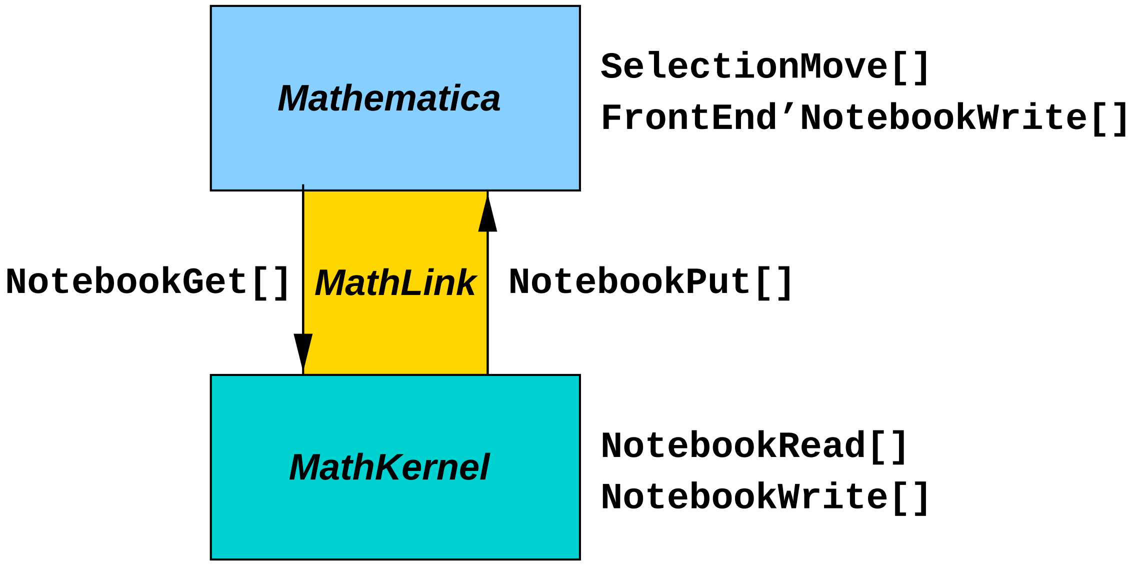
- Each notebook open in frontend represented by `NotebookObject` in *kernel*.
- Notebook content may be transferred to/from *kernel*:
  - `NotebookGet` transfers contents of a notebook to kernel.
  - `NotebookPut` updates contents of a notebook in frontend.
  - `NotebookSelection` represents current selection in frontend.
- Each frontend operation on notebook may be performed by corresponding *kernel* operation.

Notebook Kernel Operations

- Open/close a notebook cell (`SetOptions`).
- Retrieve selected contents (`NotebookRead`).
- Update selected contents (`NotebokWrite`).
- Navigate in a notebook (`NotebookLocate`).
- Context of operation may be specified:
  - `FrontEnd‘NotebookWrite` denotes frontend command.
  - `FrontEndExecute` lets frontend evaluate expression.
- Some kernel commands directly affect frontend:
  - Moving a selection (`SelectionMove`).

Kernel can control notebook in frontend!

Notebook Kernel Operations



Notebook Control

Problem: how to determine control operations issued in frontend?

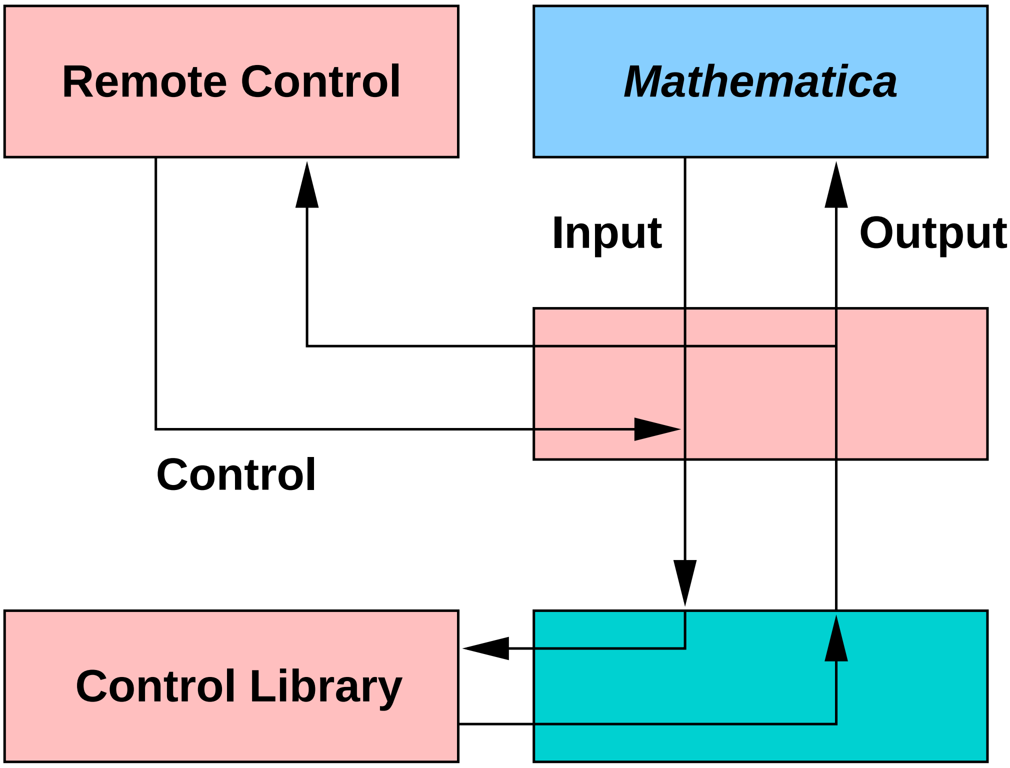
- *Mma* notebook frontend does *not* propagate control to kernel!
  - Cannot be intercepted and forwarded to other instances of *Mma*.
- Possible solutions:
  1. Write own notebook frontend.
  2. Write supplementary remote control.
  3. Embed control into notebook itself.
  4. Extend frontend by control palette.

Remote Control

- Remote control which uses *kernel* to manipulate notebooks.
- Basic navigation by control interface:
  - Open and close notebooks.
  - Open and close cells.
  - Go forward and backward between cells.
  - Propagate cell contents to kernel.
- Input/output by standard notebook frontend:
  - Evaluation of notebook expressions.
  - Directly forwarded to kernel.
- Functionality implemented in *Mma* library.

Control program may remember notebook state.

Remote Control

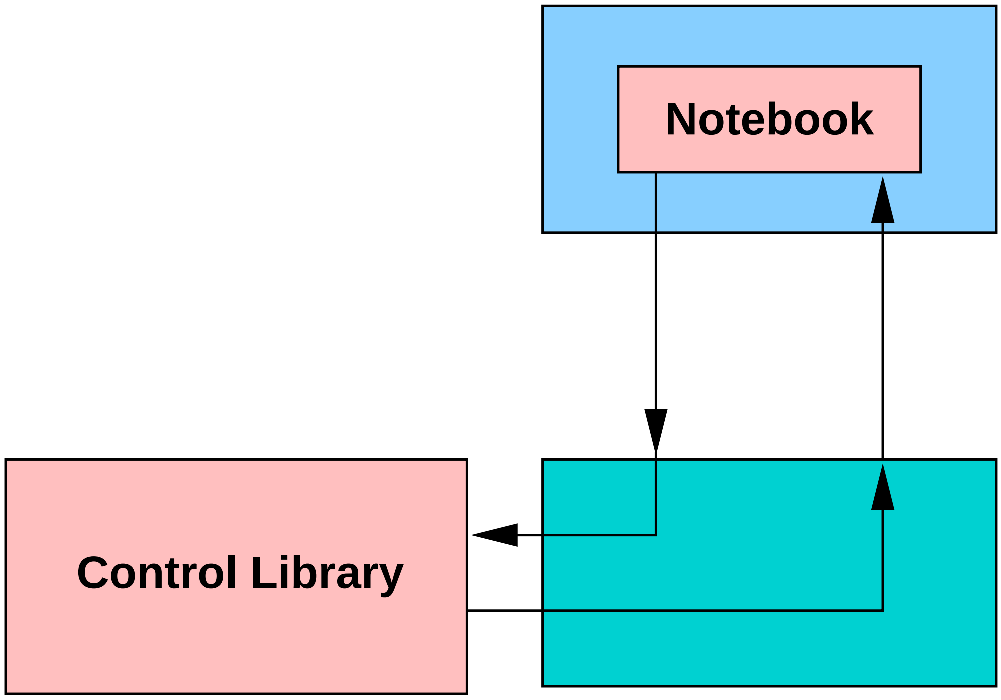


Embedded Control

- Embed control buttons in notebook.
  - Analogous to *hyperlink* buttons.
- When button pressed, expression evaluated in kernel.
  - Each control button "hardcodes" context information.
  - Command plus context transferred to kernel and evaluated.
  - Kernel updates notebook in frontend.
- Functionality implemented in *Mma* library.

Notebook may embed all control information.

Embedded Control

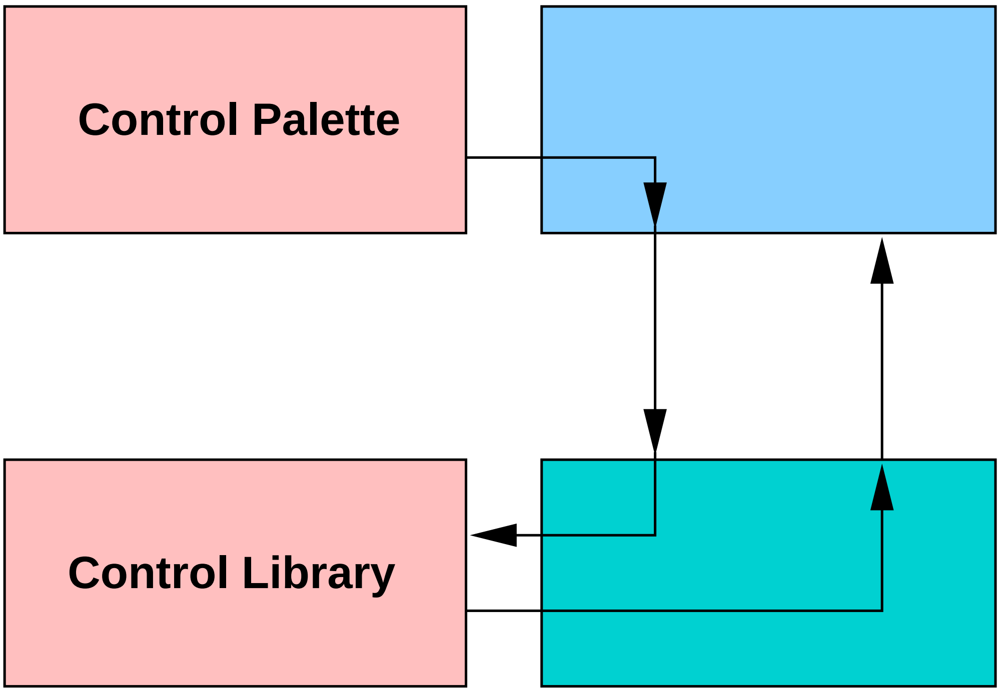


Control Palette

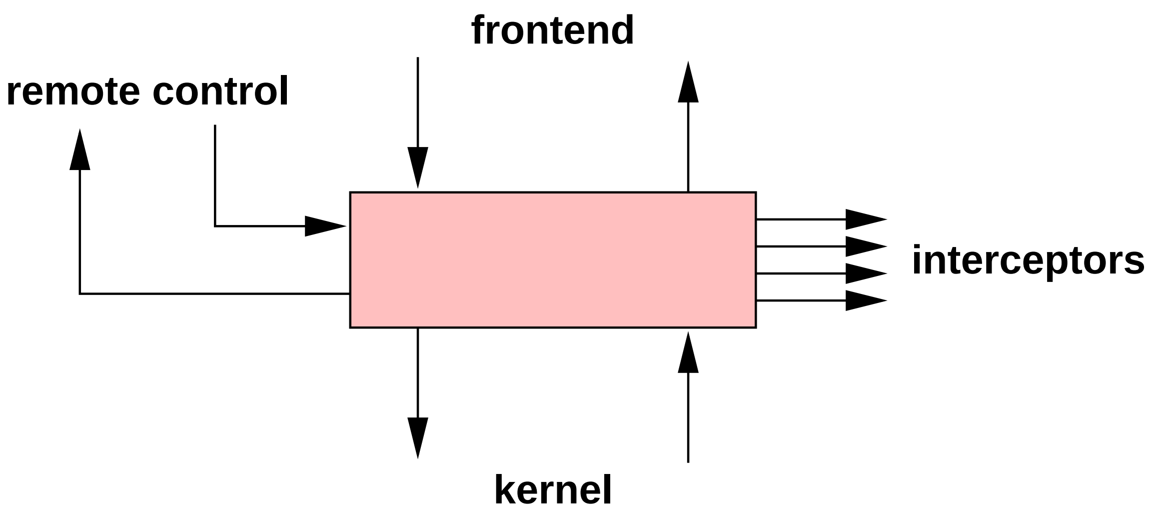
- Add control palette to notebook frontend:
  - *Palette* = menu of buttons.
- When button pressed, command evaluated in kernel.
  - Kernel remembers notebook state/position.
  - Button command updates kernel information.
  - Kernel updates notebook in frontend.
- Functionality implemented in *Mma* library.

Kernel has to remember notebook state.

Control Palette



Interceptor Process



Interceptor *transparent* to notebook frontend and to kernel.

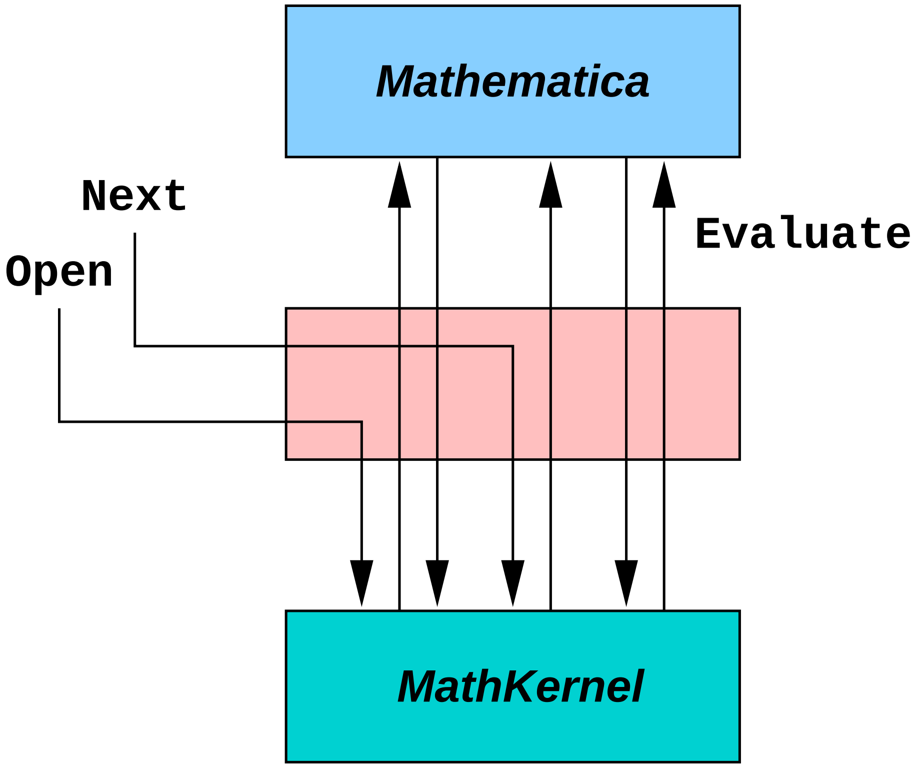
- Presents itself to frontend as a *kernel*.
- Presents itself to kernel as a *frontend*.

Local Operation

Operation in single-user mode.

- Receives *open notebook* request from remote control and forwards it to kernel which reads it and forwards it to frontend.
- Forwards navigation messages from remote control to kernel which interprets message and updates notebook in kernel and frontend.
- Forwards evaluation messages from frontend to kernel and returns results.

Local Operation



Distributed Operation

Startup phase for distributed system.

- Lecturer starts *Mma* frontend, interceptor, and kernel.
- Each participant starts *Mma* frontend and interceptor (and optionally the kernel).
- Interceptors of participants register themselves at interceptor of lecturer.
- Lecturer's interceptor subsequently *multicasts* messages to interceptors of participants.

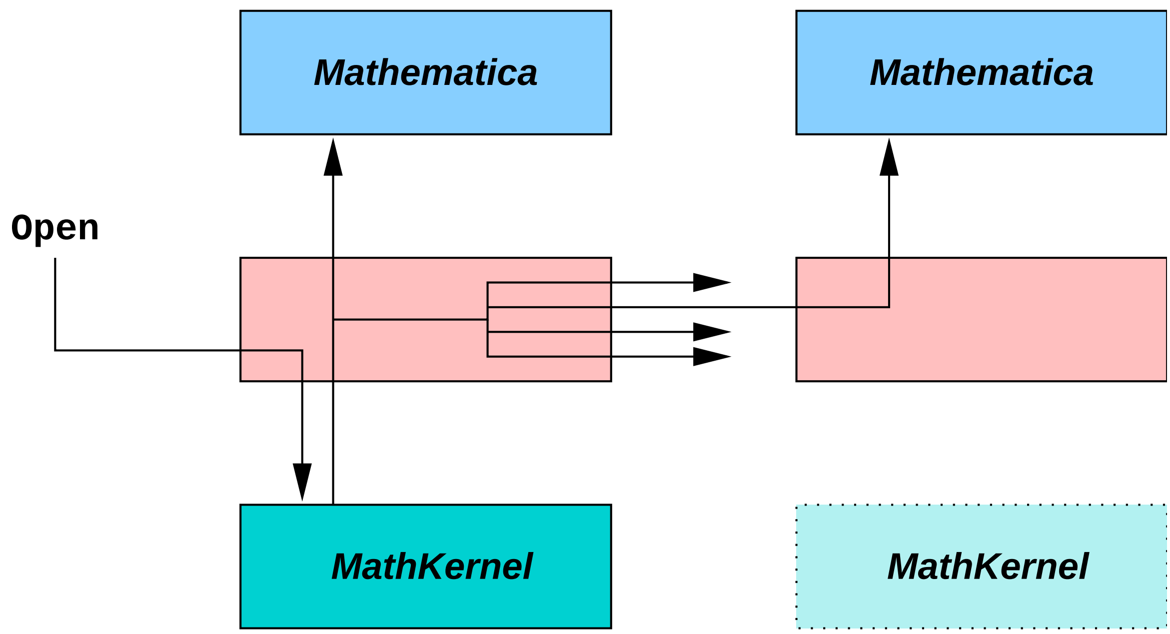
Controlled Operation

Lecturer presents notebook to all participants.

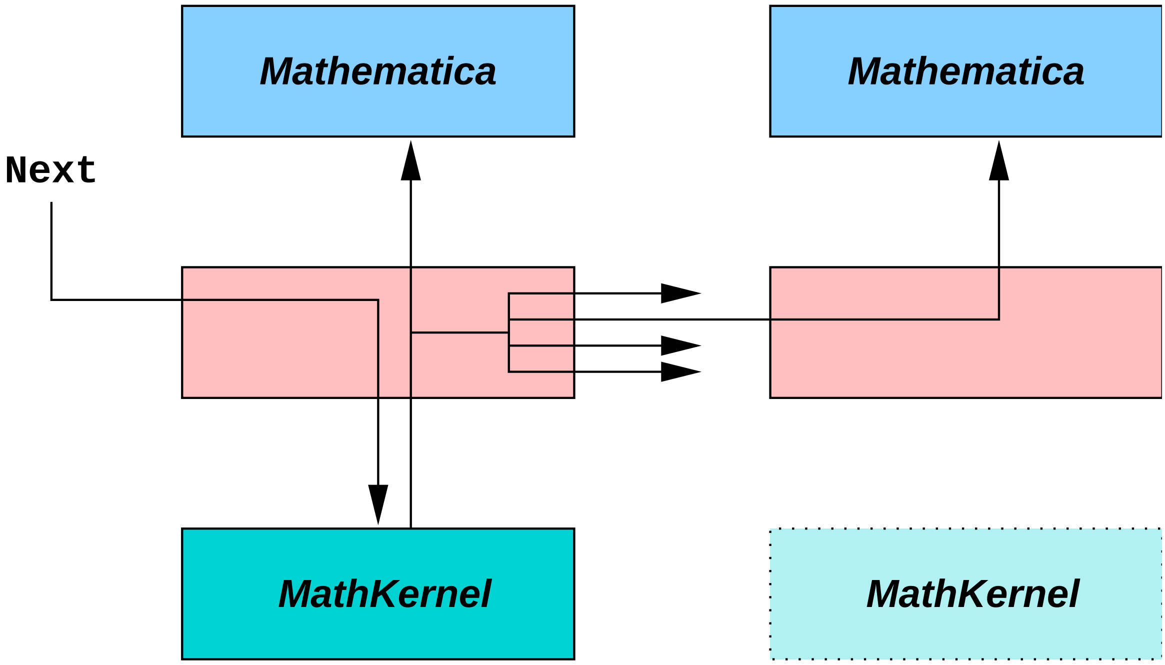
- Operation of lecturer's interceptor:
  - Handles messages from remote control locally.
  - Multicast notebooks received from local frontend.
  - Multicast messages received from local kernel.
- Operation of participant's interceptor:
  - Forward all messages received from teacher's interceptor to own frontend.

Lecturer's kernel controls all frontends.

Multicast Notebooks



Multicast Kernel Messages

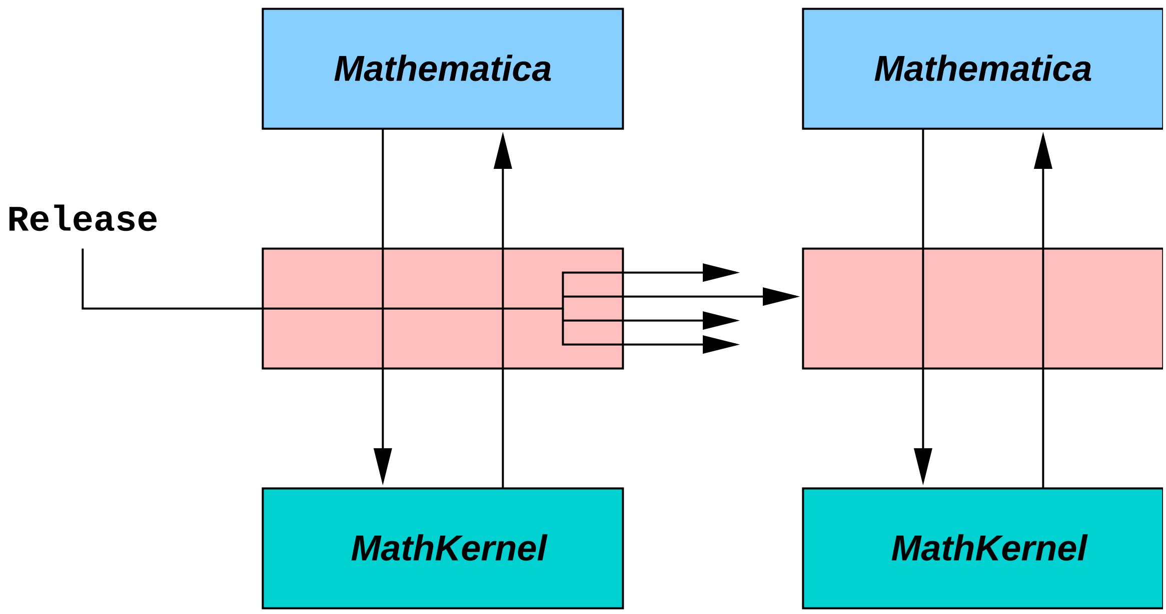


Independent Operation

Participants are allowed to perform local exercises.

- Participant's interceptor receives *release* message from lecturer's remote control:
  - Start local copy of *Mma* kernel (if not yet done).
  - Forward messages from frontend to own kernel.
  - Return results from kernel to own frontend.
- Participant's interceptor receives *hold* message from lecturer's remote control:
  - Request notebook from own frontend and save it locally.
  - (If requested) forward notebook to lecturer's interceptor.
  - Return to controlled operation (waiting for new notebook).

Independent Operation

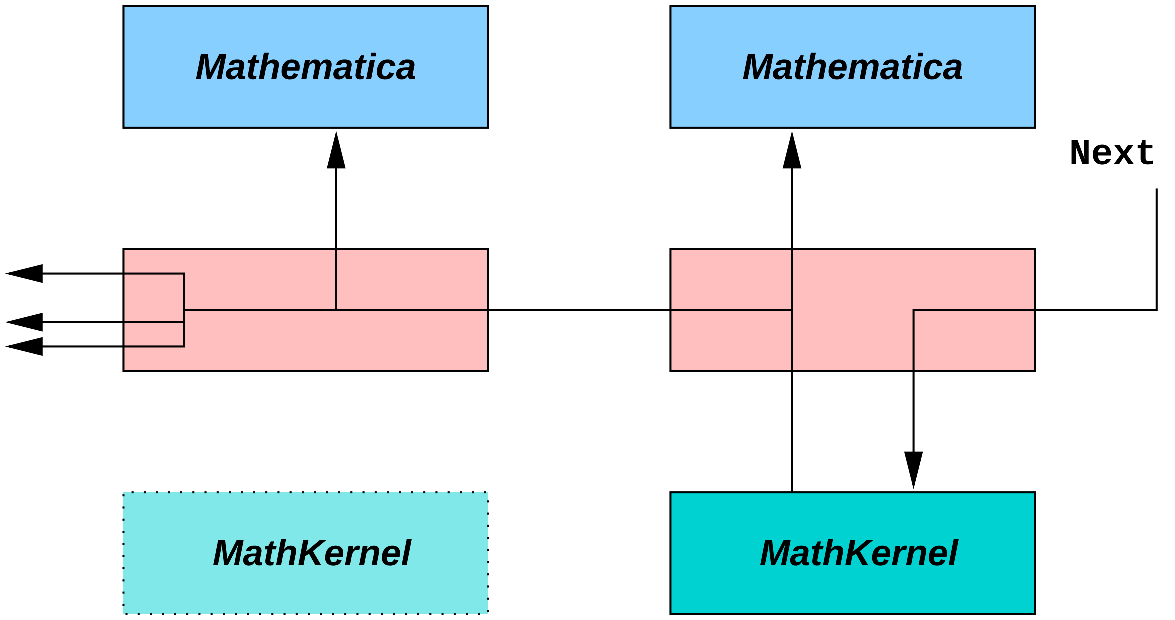


Transfer of Control

One participant is allowed to present results.

- *Transfer* message from lecturer's remote control is sent to some participant's interceptor:
  - Invoke remote control process.
  - Behavior similar to lecturer's facilties (*release*, *revoke*, *transfer* commands disabled).
  - Received messages forwarded to lecturer (for subsequent multicasting) or directly multicast to all others (*peer to peer* architecture).
- *Revoke* message from lecturer's remote control takes control back.
  - Request notebook from own frontend and save it locally.
  - Return to controlled operation (waiting for new notebook).
- Optionally notebook resulting from presentation is saved by lecturer.

Transfer of Control



Operation Cycle

1. Lecturer starts local system.
2. Participants start local systems registering themselves at the lecturer's system.
3. Multicasting of notebook.
4. Iterations of:
   (a) Controlled Operation.
   (b) Optionally one of the following:
      - Independent operation (optional retrieval of result notebooks).
      - Temporary transfer of control (optional storing of result notebook).
   (c) Multicasting of notebook.
5. Termination of session.

What about latecomers?

Multicasting

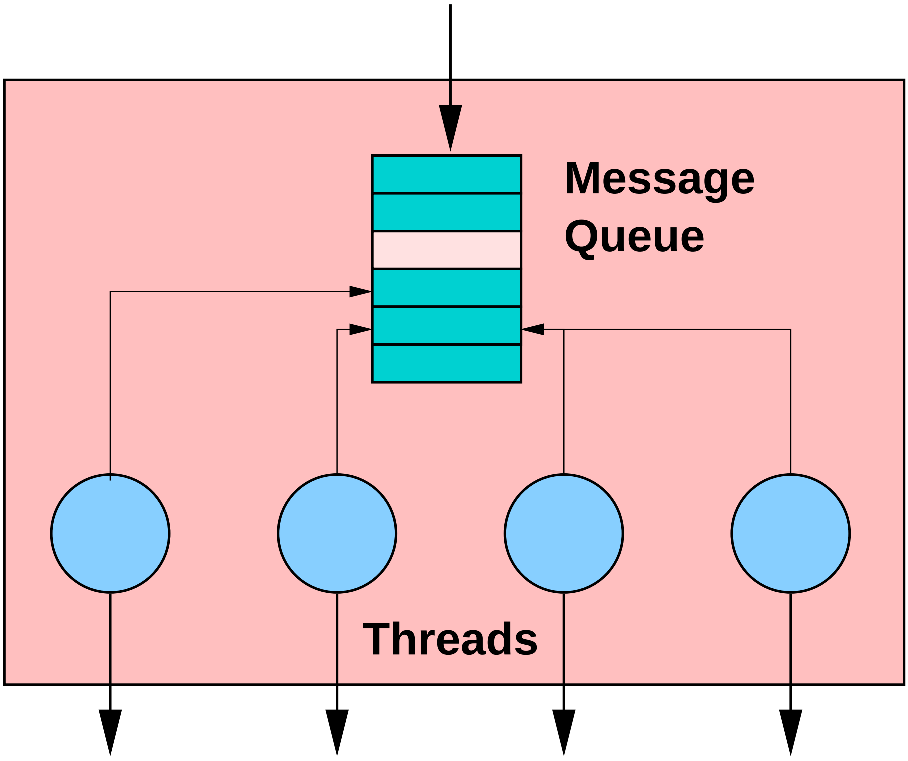
- How to implement *one to many* communication in interceptor?
  - Multicast must be reliable (inconsistencies otherwise).
  - Must tolerate slow connections to some participants.
  - Protocol overhead should be low.
- Types of messages:
  - Notebooks (large, multimedia contents).
  - Notebook inputs (medium size).
  - Evaluation results (medium to large).
  - Control messages (small).

Multithreaded Multicasting

- Multicast messages stored in single buffer.
- One thread per participant in charge of communication.
- Connection-based communication (TCP):
  - Reliable but protocol overhead.
  - Notebook contents, inputs, and outputs.
- Connection-less communication (UDP):
  - Unreliable but efficient.
  - Sequence numbering scheme to detect package loss.
  - Control events.

Independent communication with each participant.

Multithreaded Multicasting



Buffer Management

When may a buffered message be deleted?

- Arrival of a *new notebook* message.
  - Manipulations on previous notebook obsolete.
  - After new notebook, all participants in *controlled mode*.
- A message has been transmitted to all participants.
  - But what about latecomers?
  - One solution: entry allowed on next notebook only.
  - Other solution: buffer all messages until new notebook arrives.

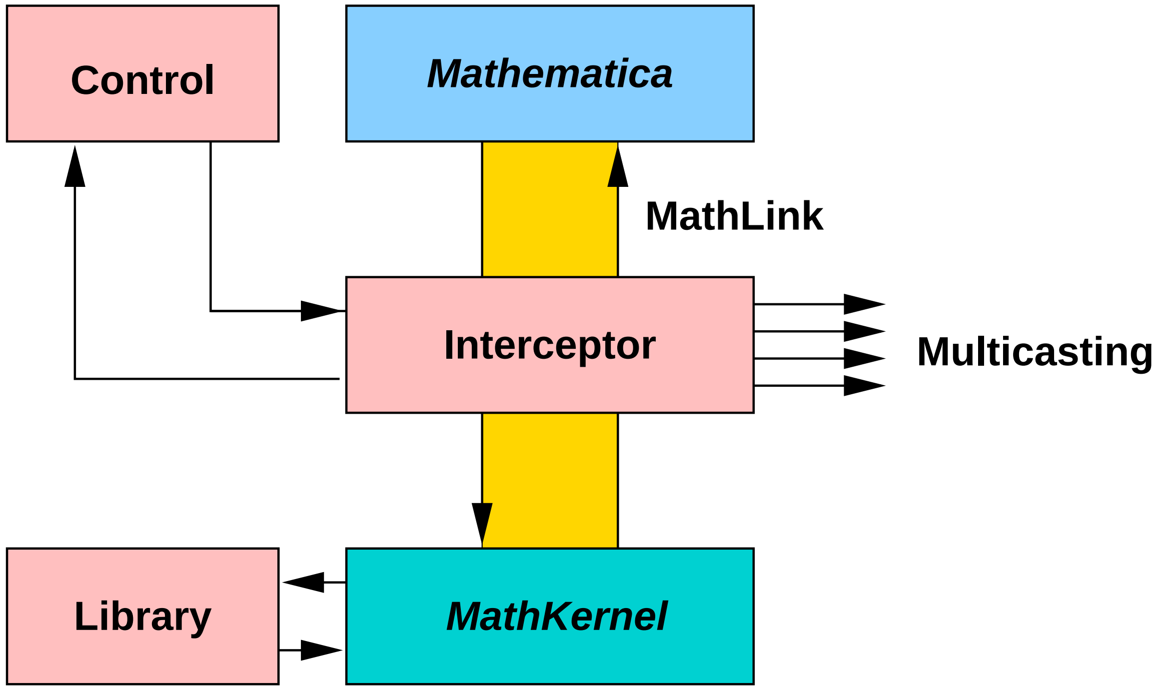
Notebook Protection

How to prevent notebooks from manipulation in *controlled mode*multikernel?

- Participants: all notebook modifications prohibited.
  - Only window scrolling (and session exit) allowed.
  - Solution: interceptors reset notebook option `Editable` when new notebook received.
  - Session exit: by local variant of remote control.
- Lecturer: navigation using frontend facilities prohibited.
  - Can't be prevented by system (???).
  - Possible remedy: display on lecturer's host also participants' view.

Putting All Together

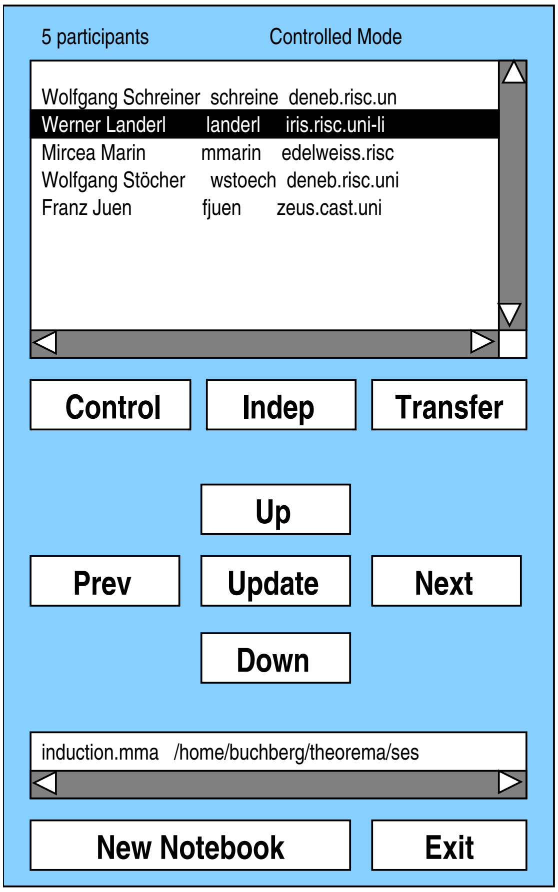
Design

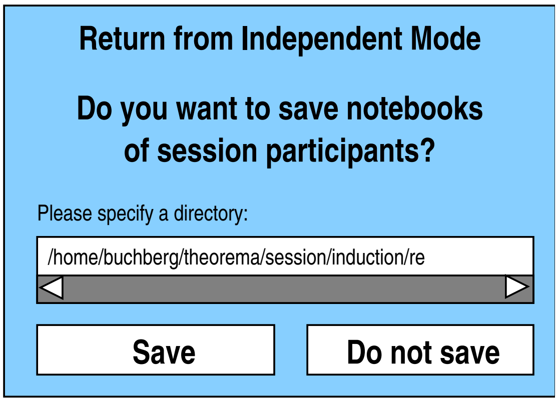

Control
Mathematica
MathLink
Interceptor
Multicasting
Library
MathKernel

Implementation

- Interceptor process in *Java*.
  - Portability, builtin multithreading.
  - *MathLink* interface by C code.
- Remote control in *Java*.
  - Portability, AWT for user interface.
  - For lecturer and participants.
- Supplementary library in *Mma*.
  - Notebook manipulation.
  - Frontend control.

User Interface





Conclusions

- *Mma* as a component of a distributed education system.
- Frontend and kernel *unmodified*.
- Functionality by interceptor, remote control, and supplementary *Mma* library.
- Diploma thesis:
  - Implement interceptor process to *monitor Mma*'s communication.
  - Experiment with embedded control and control palette.
  - Develop remote control for single-user mode.
  - Extend system to multi-user mode.
  - Develop control transfer.
  - Develop independent execution.
  - *Revert design decisions.*

Develop suitable session contents!