List of Slides

17 Command line user interface
18 ARCADE commands
19 ARCADE commands continued
20 ARCADE commands continued
21 ARCADE commands continued
22 ARCADE commands continued
23 ARCADE commands continued
24 Command extensibility
25 Example: send info to other department
26 Example: send info to other department
27 Example: send info to other department
28 Automatic tasks
29 On line help
30 Efficient data entry interface
31 Trusted user GUI
32 Trusted user GUI screen shot
33 GUI specification language
34 Example GUI specification: `stud` command

Case study 3

ARCADE

A small system

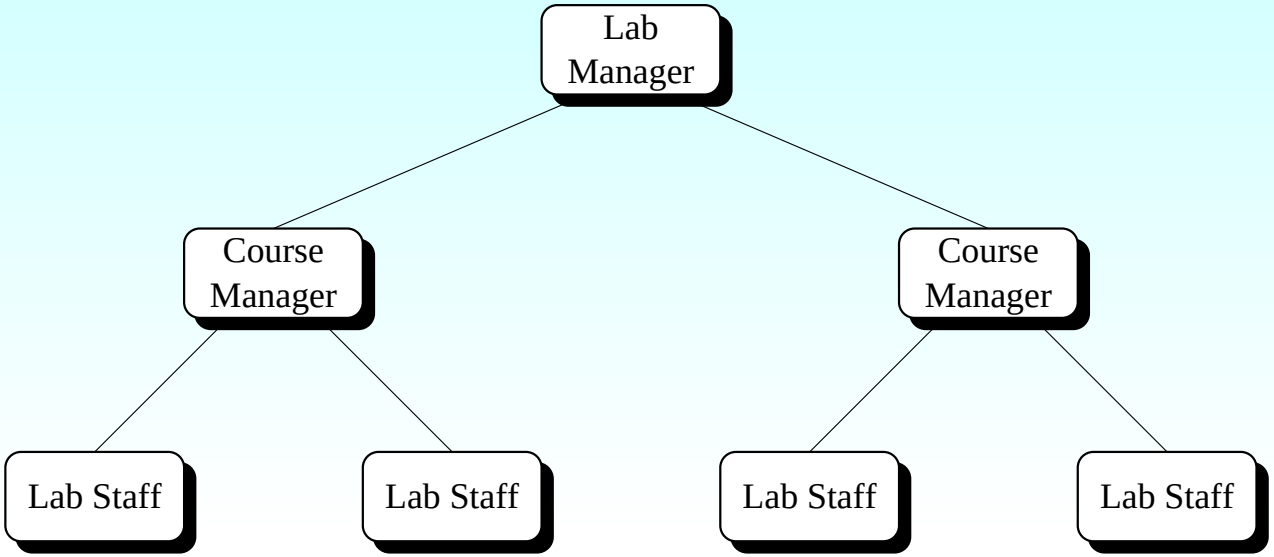
Why study ARCADE?

- All the examples you have seen throughout CS1081 and CS1092 have been small single program applications.
- The programs in the laboratories have mostly been even smaller.
- It is important for you to begin to put your experience into a wider context – real applications are much bigger, and there are other issues than just code development.
- A **system** consists of a collection of cooperating programs, persistent data and external entities such as users and other systems.
- ARCADE is such a system, and because you are already (partly) familiar with it, and (should have) a vested interest in it, it is a convenient and motivated example!

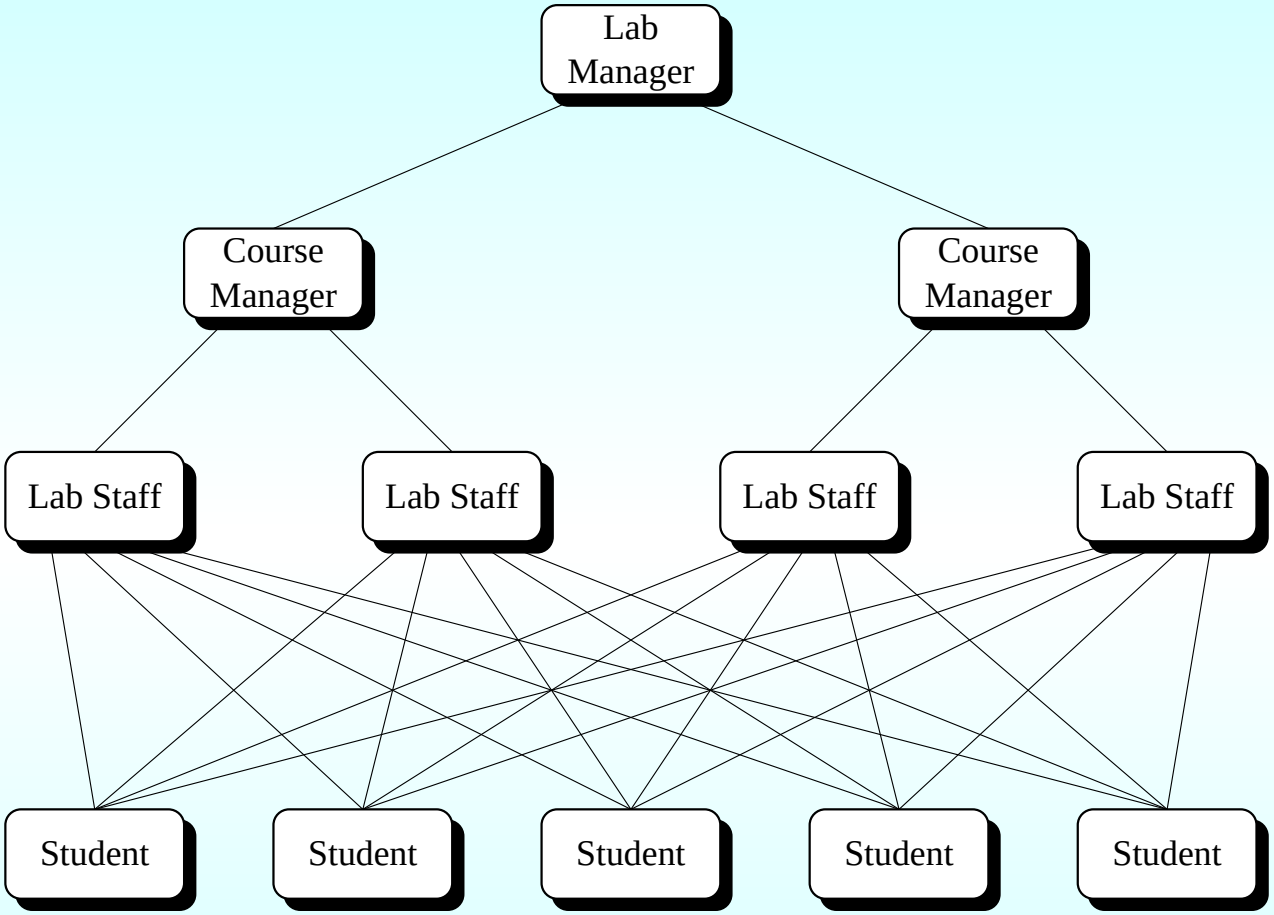
What is ARCADE for?

- ARCADE is a management system.
- Managing Human behaviour is typically a challenging area of computer application, because Humans do not behave predictably.
- Many Human management systems operate on a simple pyramid structure, and the smooth operation is merely as good as the chain of command.
- Many people thought that managing a year of laboratories would involve such a pyramid structure.
- However, they were missing something very important from their view which makes the structure significantly more complicated.

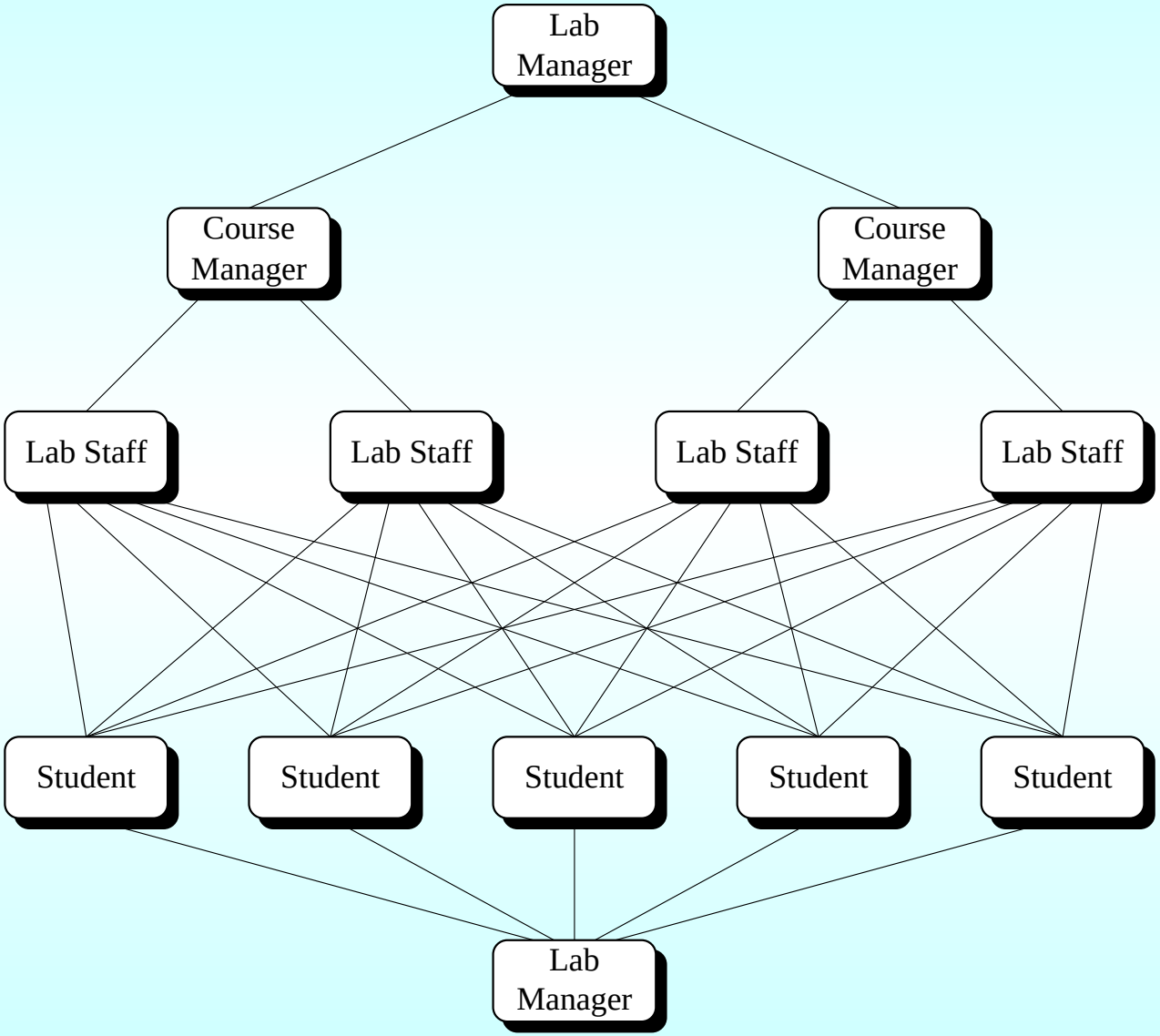
ARCADE people interaction: myth



ARCADE people interaction: reality



ARCADE people interaction: desire



Brief history of ARCADE

- ARCADE was born in September 1993: I was asked to manage the first year laboratories under the new semesterised and modularised syllabus.
- A cause of concern was that the second year laboratories already had such a structure, and had been horribly chaotic for several years.
- The inability to easily manage deadlines in such a fragmented structure meant there was only one deadline for each course – at the end of it.
- For the first two thirds of the first term, the demonstrators would actually be sat playing cards and drinking coffee, as there were so few students wanting any help.
- Then for the last few weeks, there would be a mad rush of panic: marking queues more than 3 hours long and a 40-strong delegation of students invading the head of UG's office, demanding that the "too tight" deadlines were extended into the second semester.

Development process

- As is typical of many systems, ARCADE was developed rapidly at first, slowing to a trickle later.
- The development process has been incremental, using the **spiral model** of system development.
- This approach spirals outwards around a central starting point, going through phases such as **requirements analysis**, **specification**, **design**, **implementation**, **validation**, etc..; at each cycle getting further away from the central point as the system gets bigger. More about that in Software Engineering next year.

Development process

- Features were added to ARCADE in a demand-driven way: the first task was to identify the philosophical structure (e.g. deadlines with extensions, etc..); then produce software which could print paper forms so that data could be recorded, then develop a program to make it easy to record that data, then software to process it, etc..
- Later came software to provide email feedback to students and tutors.
- At first laboratories only happened on dates – times were added later.
- The generation of timetables was a feature added later still.
- The ability to specify dates by semester number, week number and day of the week was added after a couple of years – this seriously helps with programming a laboratory structure.
- Much later came the client query software.
- Etc.. This is typical of most real world systems.

A legacy of languages

- In the beginning ARCADE was developed using the language **Pascal** together with **shell script**s.
- (Aside: Pascal was chosen rather than the obvious alternative, **C**, because of personal preference, but also to prove a point: many people viewed Pascal as a teaching-only language, and not one suitable for real world programming.)
- The system was developed on Sun equipment, using Sun Pascal.
- To port it to Linux a Pascal compiler was created (based on **p2c** – a program which attempts to turn Pascal into C).
- More recently, development has included the language **Perl** and **Java** (the latter for the client query service).
- Such a mixture of languages is typical of real world systems.

Size of ARCADE

- ARCADE is a *small* system. (Note: ARCADE does not include **labmail**, **labprint**, the plagiarism detector nor the course choices web form. They are not strictly part of ARCADE.)
- ARCADE currently comprises

26 Pascal programs	19162 lines of code
28 Pascal modules	15944 lines of code + 1695 in header files
135 shell scripts	11671 lines of code (excluding built-in help and GUI descriptions)
3 Perl scripts	6262 lines of code
24 Java files	3550 lines of code
3 Makefiles	1195 lines
Total lines of code	59479

What is atypical about ARCADE?

- There are two main features of ARCADE's development that are atypical of real world systems.
  - It has been developed by just one person rather than a team of people changing over time.
  - The developer has also been the customer.
- This has made the development of ARCADE much easier than most real world systems.

Requirements Analyisis

- System development starts with **requirements analysis**. This can be a lengthy and complex process, and lack of attention to it has often led to disaster later (e.g. LAS).
- To fully describe requirements of something as simple as ARCADE would need a 30 page document.
- Approx 250 students, approx 7 lab. groups, approx 40 tutorial groups.
- Tut group is often a sub-set of lab group, but does not have to be. (E.g. 2nd year labs have no relationship to tut group.)
- E.g. lab groups: B=C+D, M=P+Q, W, X, Y, Z.
- Approx 5-10 sessions per course (variable) per group.
- Sessions are associated with exercises, some with deadlines.
- E.g. sessions 1D, 2.1, 2.2D, 3D, etc..

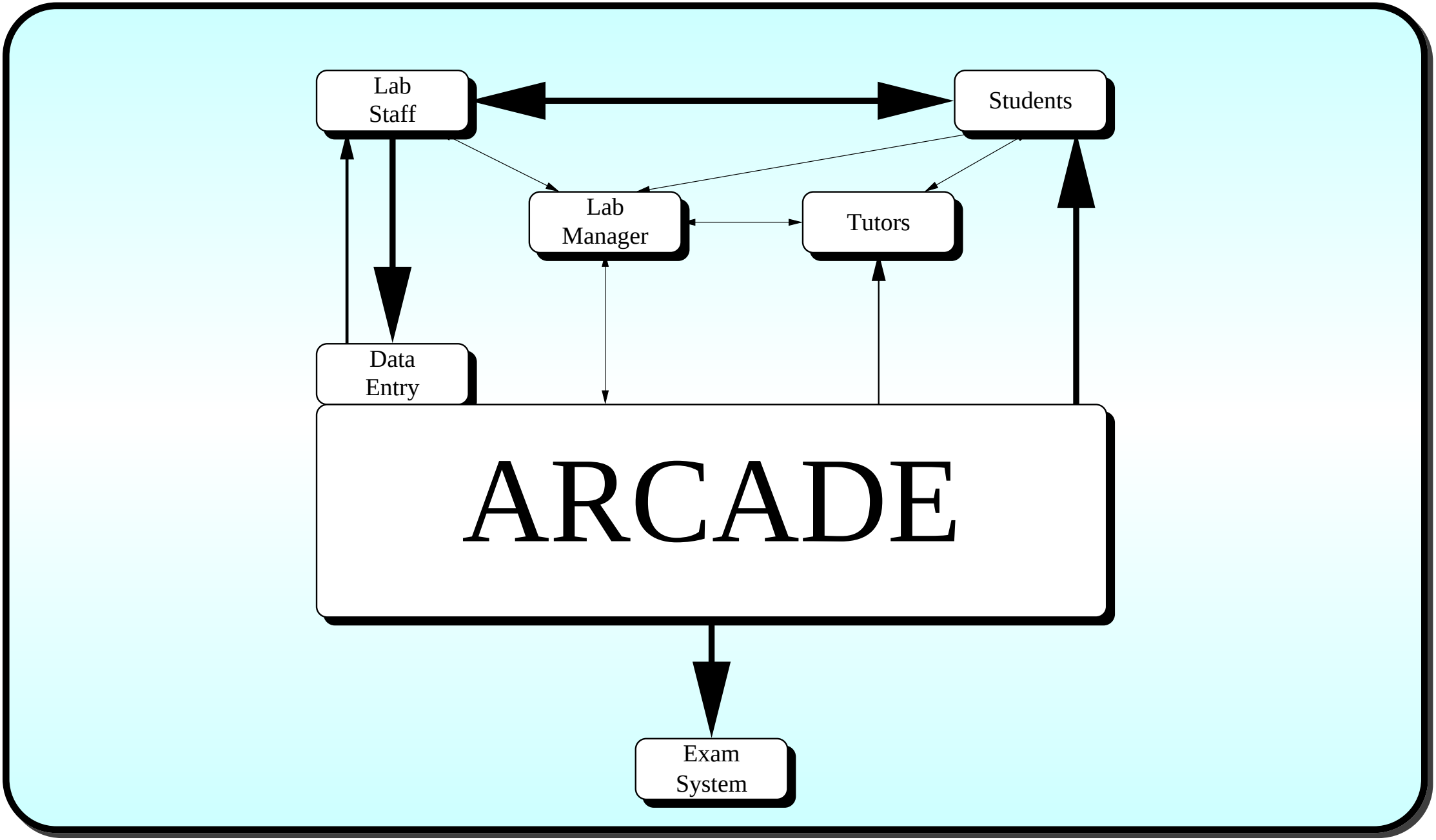
Requirements of ARCADE continued

- Attendance at sessions is usually monitored.
- Sessions each have scheduled date, time and place for each group.
- For deadline sessions, deadline is at the end of the session, but may usually be extended to extension deadline, which is usually the start of the next session.
- Deadline sessions have marks associated, marks per deadline per student. They have a maximum mark, bonus marks, a denominator and a weighting. Weighting may be optional – i.e. the session does not count if the student does not do the work.
- A deadline session can be a reference to another course – it takes at least a page to explain this properly!
- And so on. This has barely scratched the surface.

ARCADE users

- There are currently 3 classes of users involved with ARCADE: **trusted** users, **standard** users and **laboratory demonstrators**.
- The trusted users are those few people who have full access, and include year laboratory managers, year tutors, data entry clerk, and so on. These users have various ARCADE permissions to perform various tasks, but such permissions operate on a **cooperative security** scheme. That is, they serve to protect against accident rather than deliberate abuse; by tailoring the user interface(s) to offer only the tasks interesting to the person.
- The standard users are students, tutors and laboratory supervisors, who are entitled to use the remote query service. These people have no direct access to ARCADE for obvious security and privacy reasons.
- The laboratory demonstrators have access to ARCADE only via the paper data sheets.

ARCADE context dataflow diagram



Command line user interface

- ARCADE was developed at first with only a command line interface.
- This reflects a philosophy typically found in the non-commercial Unix world: concentrate on getting the functionality right before deciding on the user interface.
- Which is in contrast to the philosophy often found in the commercial, especially Windows, world: make it look posh and attractive, and figure out what it should do later! :-)
- However, the original ARCADE command line interface did exploit **command name completion** to save typing. An ARCADE database is a Unix directory, and all the ARCADE commands are available as symbolic links in that directory. So the command name completion facility of `ksh`, and later `bash`, meant the names could be long and meaningful, but still easy to type.

ARCADE commands

- There are currently about 80 command in ARCADE. Each of these performs a high level task of the system.
- ARCADE commands are all shell scripts, which call other programs as necessary.
- Many of them are listed here to give you an idea of the wide range of tasks involved.

```
absence-list.........................show non-attendance details
absence-summary.....................show attendance summary
add-courses.....................................add new modules
arcade..............start the ARCADE graphical user interface
attendance.........................send attendance data by email
bad.................................show 'bad' irregularities
blank...............................show where data is needed
checking..............report on or switch data checking status
```

ARCADE commands continued

```
completion-graph.............................show completion graph
config-mail-bad-summaries
 ..............configure email feedback to students (and tutors)
config-mail-full-stories
 ......................configure email full stories to students
config-mail-student.........................configure student mail
course-summary..............show summary statistics of a module
data.....................................enter or browse data
doing........................show numbers of students on module
due.....................................show what needs to be done
edit-privileges........edit user privileges for current database
edit-sheet-details...edit sheet-details describing lab structure
edit-sort-groups-file.........................edit a groups file
email..............send email to a group / selection of students
email-name-check.........................check student email names
excuses.....................................show excuse details
```

ARCADE commands continued

```
expected.....................................show expected work
fails.............................show students who are failing
final-diffs.....show differences between two sets of final marks
full-story.....................show full lab details for a student
group-info
.........show module group, students and staff hours information
help.....get help on ARCADE commands or UNIX commands/facilities
histogram.............................show histogram of a module
licence.....................................display end user licence
list-all-modules
............show names of ALL modules taken by 1 or more students
list-arcade-modules..................show names of ARCADE modules
list-table-modules..show names of ARCADE modules having deadlines
mail-bad-summaries
...........send automatic email feedback to students (and tutors)
mail-full-stories.................email students full lab records
```

ARCADE commands continued

```
mail-name-list.....................email student details to students
mail-report
.......email a report to the students in it, with an introduction
mail-sessions-table............email student timetable to students
make-email-script
......make an email script for a module (and group), or selection
make-final-marks
........make final (or predicted) marks for all completed modules
marking............list sessions which have marking or demo needed
name-list-compare.....compare student details with external files
name-list-edit
...............add new students or edit existing students details
name-list-trace...............trace changes to a student's record
name-list-update
....create, update or compare student details from external files
omitted.......................show where data is (probably) missed
```

ARCADE commands continued

```
out-of-range..............list sessions with out-of-range marks
outstanding................................show oustanding details
picture....................................show student pictures
print-course-summaries.....................print course summaries
print-full-stories.................print students full lab records
print-histograms...........................print module histograms
progress...................report simple overview of all modules
queries....................list sessions with unanswered queries
read-mail..................................read email from a student
remote-jobs-control....set up jobs to be done remotely by crontab
restore-all-data...........................restore data from back-up
scaling.............................show scaling factors for modules
select-and............................intersect two selection files
select-copy................................copy a selection file
select-edit................................edit a selection file
select-group............................select a group of students
```

ARCADE commands continued

```
select-list.............................list an existing selection
select-minus.........................difference two selection files
select-move.................................rename a selection file
select-or.................................union two selection files
select-remove.............................remove a selection file
select-show.............................show names of selection files
select-specific.........................make a selection of students
sessions-table..............show sessions timetable and attributes
sheets.....................................print session sheets
stud.......................................show student details
supervisors-report.....show ‘bad’ irregularities for a lab-group
table..................show tables for certain students or modules
unchecked.........................list sessions with unchecked data
```

Command extensibility

- Command line interfaces are inherently more easy to extend than any other kind of interface.
- It can be trivial for the (expert) ARCADE user to create new ARCADE commands by combining existing ones with standard Unix tools in a shell script.

Example: send info to other department

```
#!/bin/sh

COURSES="1581L 1582L 1580E"

MAILTO=<<list of recipients has been deleted from slides>>

( echo "Attendance summary for $COURSES"
  echo "================================"
  echo "    (/ = present, x = absent, E = excused, ? = data is due)"
  echo
  ./absence-summary -s ABIS -t 100 $COURSES | ./stud - -d -Name
  echo
  echo
```

Example: send info to other department

```
echo "Marks table for $COURSES"
echo "========================"
echo "           (n) is a predicted mark"
echo "           E means work is expected in the future"
echo "           L means work is late"
echo "           - means work has been missed"
echo
./table -s ABIS -d $COURSES
echo
echo
```

Example: send info to other department

```
  for course in $COURSES
  do
    echo "Irregularities for $course"
    echo "=========================="
    ./bad -s ABIS $course | grep -v chived \
      | ./stud - -Name \
      | ../EXECUTABLES/tidy-report
    echo
  done
) > TEMP/abis-summary.txt

unix2dos TEMP/abis-summary.txt

mpack -s "ABIS progress" TEMP/abis-summary.txt $MAILTO
```

Automatic tasks

- ARCADE has a facility to perform automatic tasks, e.g. nightly at 3am.
- Any Unix commands can be conditionally run, and the output logged in a temporary file.

```
#Thursdays
?test ‘/bin/date +%w‘ = 4 -o ‘/bin/date +%w‘ = 4
../EXECUTABLES/mail-bad-summaries

?/bin/true
../EXECUTABLES/attendance
../EXECUTABLES/mail-name-list | fgrep -v "same as last time"
../EXECUTABLES/mail-sessions-table | fgrep -v "same as last time"
```

On line help

- Commands have user help built into them.
- These are stored in the shell script as shell comments and are pulled out via a help program. For example the `stud` command contains:

```
#!/bin/sh
...
#man
#This program shows extracts from the STUDENT-DATABASE/name-list
#file. It can show a subset of the students, and/or a subset of
#the fields.
#...
#end-man
...
```

- This approach is convenient, and encourages the help text to be immediately maintained whenever the command is altered.

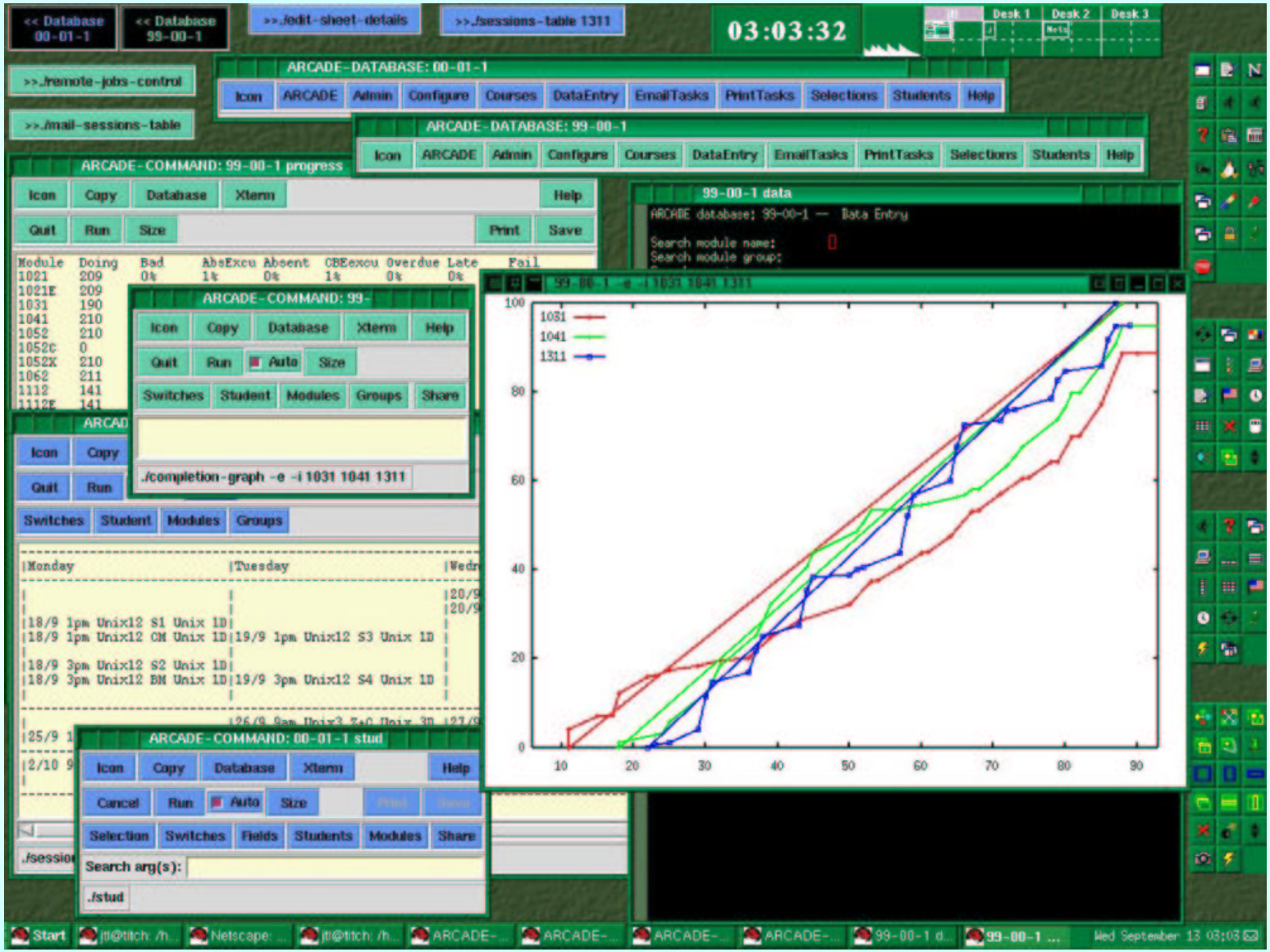
Efficient data entry interface

- One of the early design principles of ARCADE is that it must be reasonably efficient to run within the available resources.
- A key focus in this respect was on the user interface for the data entry program: all data from the laboratory sessions was to be hand entered from written sheets.
- This meant designing the data sheets carefully in the first place.
- And providing an economic single-key oriented data entry program.
- This program has a voice prompting facility – it reads out the 3 digit student identifier of the current student when one is selected. This means a data entry user does not have to look at the screen to know which student's data s/he is editting.
- Also, data in the various fields is inferred from data in other fields where possible, so that typing is minimised.

Trusted user GUI

- After seven years of not needing one(!), a GUI was developed in 2000, using Perl/Tk.
- The design of this had in mind the non-expert trusted user, but it has turned out to be useful for the expert trusted users as well.
- E.g. I tend to use the command line interface for doing just one or two things, or the GUI for doing several things.
- This GUI enables the user to set up many instances of ARCADE commmands and leave them permanently on his or her ARCADE ‘desktop’. Thus it behaves a little like a window manager.
- The GUI sits on top of the command line interface.

Trusted user GUI screen shot



GUI specification language

- The GUI is extensible. It actually interprets a tailor-made GUI specification language at run time.
- The GUI specifications are embedded in the shell scripts which make up the ARCADE commands – as shell comments.
- When it opens a database, the GUI program examines which commands the user is allowed to run, and places them on the command launching menus as specified in the source of the commands.
- When it opens a command, the GUI reads the source and builds a command tool according to the GUI specification in the command.
- This way, it is possible to add new ARCADE commands without needing to alter the GUI program in any way at all.
- It is even possible for the user to write GUI specifications for the commands he or she has built from existing ARCADE commands.

Example GUI specification: `stud` command

```
##GUIMENU=Students

##GUISORT=yes
##GUISAVE=yes
##GUIPRINTER=yes
##GUIAUTOUPDATEON=yes
##GUISELECTION=yes
##GUISWITCHSET=yes

##GUISWITCHARG=-a
##GUISWITCHFULLNAME=Expand email addresses
##GUISWITCHTYPE=none
##GUISWITCHVALUE=
##GUISWITCHEXCLUDESSELECTION=no
```

Example GUI specification: continued

```
##GUISWITCHARG=-cs
##GUISWITCHFULLNAME=Create a selection from output
##GUISWITCHTYPE=string
##GUISWITCHVALUE=stud-out
##GUISWITCHEXCLUDESSELECTION=yes

##GUISWITCHARG=-d
##GUISWITCHFULLNAME=Neat display
##GUISWITCHTYPE=none
##GUISWITCHVALUE=
##GUISWITCHEXCLUDESSELECTION=no
```

Example GUI specification: continued

```
##GUISWITCHARG=-l
##GUISWITCHFULLNAME=Include left and unregistered
##GUISWITCHTYPE=none
##GUISWITCHVALUE=
##GUISWITCHEXCLUDESSELECTION=no

##GUISWITCHARG=-i
##GUISWITCHFULLNAME=Case insensitive search
##GUISWITCHTYPE=none
##GUISWITCHVALUE=
##GUISWITCHEXCLUDESSELECTION=no
```

Example GUI specification: continued

```
##GUISWITCHARG=-t
##GUISWITCHFULLNAME=Tab separate single match
##GUISWITCHTYPE=none
##GUISWITCHVALUE=
##GUISWITCHEXCLUDESSELECTION=no

##GUIFIELDNAMES=yes
##GUIFIELDNAME=-StudentId
##GUIFIELDNAME=-Registered
##GUIFIELDNAME=-RegNo
##GUIFIELDNAME=-Degree
##GUIFIELDNAME=-Year
##GUIFIELDNAME=-Owner
##GUIFIELDNAME=-LabGroup
##GUIFIELDNAME=-TutGroup
##GUIFIELDNAME=-Tutor
```

Example GUI specification: continued

```
##GUIFIELDNAME=-Name
##GUIFIELDNAME=-DBSurname
##GUIFIELDNAME=-DBFirstNames
##GUIFIELDNAME=-Email
##GUIFIELDNAME=-Modules

##GUISEARCHSTRING=yes

##GUIMODULESET=yes
##GUIMODULEFILTER=direct
##GUIMODULEPREFIX=\"
##GUIMODULEPOSTFIX=( |$)\"
```

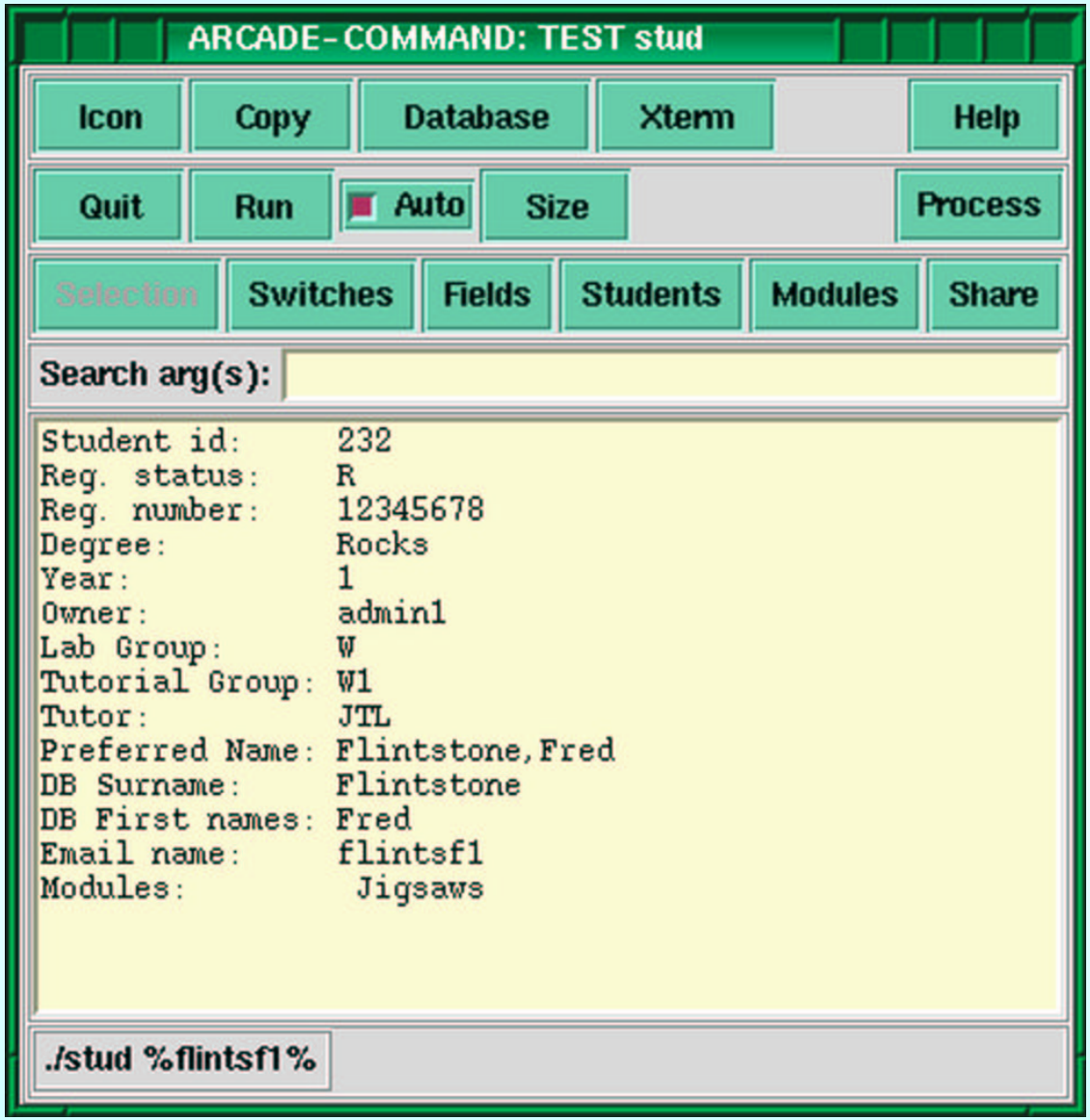
Example GUI specification: continued

```
##GUISTUDENTSET=yes
##GUISTUDENTPREFIX=%
##GUISTUDENTSEPARATOR=\|
##GUISTUDENTPOSTFIX=%

##GUISELECTIONEXCLUSIVEWITHSTUDENT=yes

##GUISUBMISSIONOUTDATESSELECTIONS=yes
```

stud command GUI



Standard user remote query service

- A later development was the remote query service for standard users.
- This is provided by a server program, written in Java, communicating with a client program, also written in Java.
- This is known as a **client-server** model. There is one instance of the server program running somewhere, and many instances of the client program, running on different computers, talking to the server.

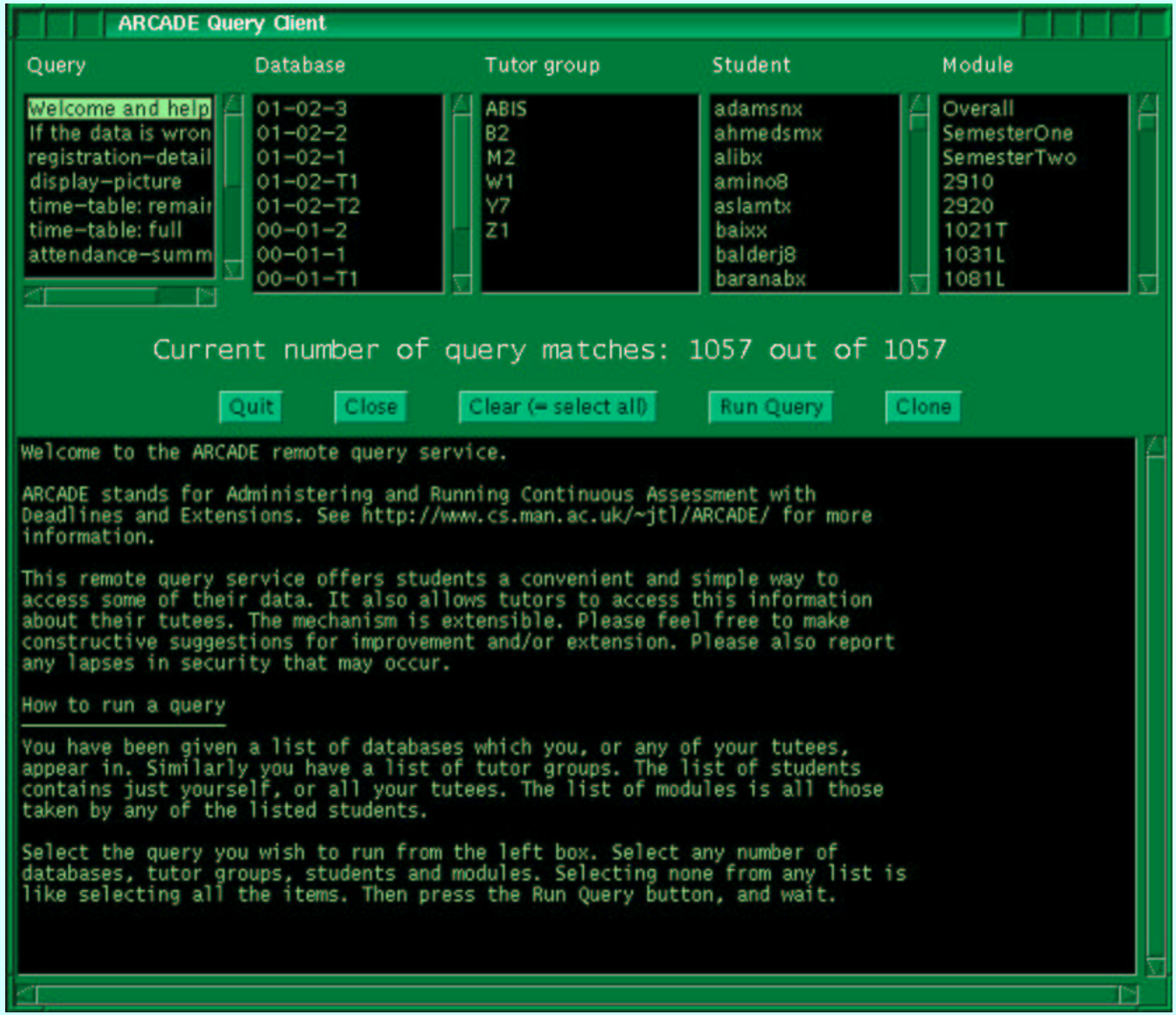
Query service server

- The server program consists of just over 500 lines of Java code – quite small.
- It sits on top of a shell script which sits on top of (a copy of) ARCADE.
- The server checks authenticity and permissions of the users trying to connect to it.
- The server is **multi-threaded** so it can serve many transactions at once. It has a configurable limit to the number of connections at any time.
- The client has to authenticate itself to the server whenever it connects.
- Each connection is for one transaction only, i.e. a single run of a single query from a client. Why is this good?

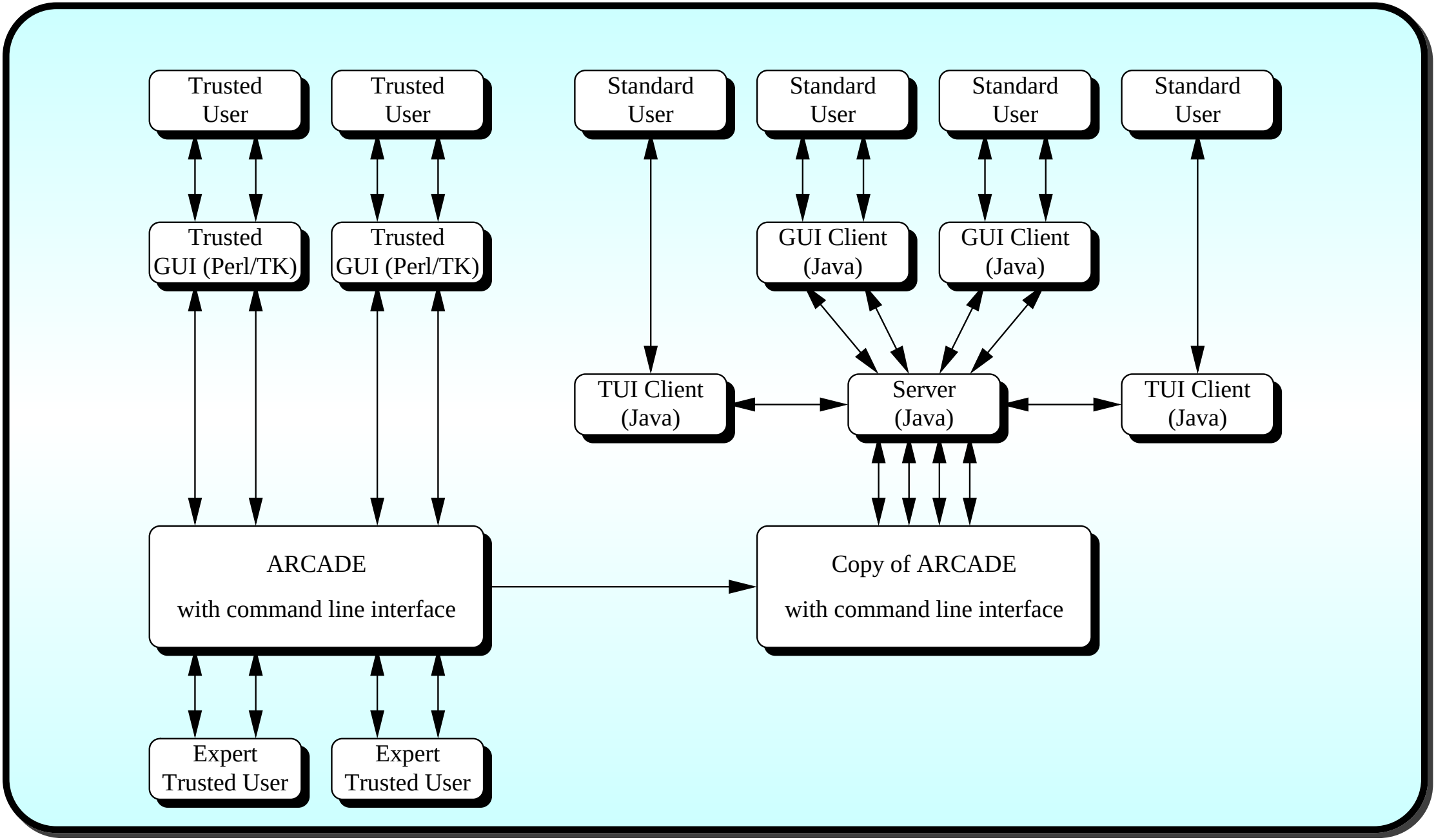
Query service client

- The client program consists of a little more than 2500 lines of Java code – quite large by comparison. Why?
- It offers various **canned queries** with arguments.
- Queries and argument profiles are obtained from the server at run time. The server obtains these from ARCADE, for the user of the client. This means the whole interface is fully extensible without any need for changing the Java code.
- The client dynamically adjusts the argument lists to match the user's profile – as the user selects items, options in other lists are cut down.
- The client can be cloned so the user can run more than one query at once. The client is multi-threaded so it literally can run more than one query at once.
- (A single-threaded text interface Java client is also available.)

Standard user client GUI



ARCADE architecture



Plan for errors

- The behaviour being managed by ARCADE is fundamentally chaotic and error prone.
- ARCADE imposes a structure which helps reduce this chaos.
- But we must expect a certain level of Human error.
- ARCADE is designed on the 95-5 principle: 95% of the work of the system will go smoothly, 5% will need extra interaction.
- Actually it is more like 80%-15%-5%.

Errors compound

- Once one thing has gone wrong it becomes more likely that others will.
- For example, a student misses a lab through illness.
  - The student is now in a (relatively) unsual situation for that lab, and so the error likelihood increases dramatically.
  - The student might tell the lab staff the reason for his or her absence.
  - One possible error is that he or she tells a demonstrator rather than the supervisor, and thinks it is all sorted.
  - Or the student tells the supervisor, but the supervsior forgets to write it down. Or the supervisor writes it in the wrong place.
  - Perhaps the student is now a little behind, so his or her work is marked from an earlier exercise than everyone else at that time. So it is more likely than usual that the mark might get recorded against the wrong exercise. Etc..

Expected error rate

- Given the pressure in the labs, and the large number of people involved in the process, it is reasonable to expect most students will have at least one error made in their data during the year.
- If a student is ill for a significant period then it is very likely that many errors are made in sorting out excuses etc..
- This is the nature of errors and exceptional behaviour. Systems need to be able to cope with this reality, by having checks etc..
- E.g. ARCADE expects students to check their own data ....

The future

- Although development is now much slower than in the beginning, ARCADE will continue to evolve until such time as it is replaced by some other system.
- This is typical of real world systems: software lives forever and continues to develop.
- One of the plans for the near future is to reduce some of the error probabilities – e.g. on-line mark submission might be in place next year ....