
USING Computer Science

Applications:

- Concurrency – More than one activity.
- Multimedia and Hypermedia – Huffman encoding (GIF)
- Artificial Intelligence – Turing test.
- Cryptography – hiding information.

Multimedia

Multi-sensory, interactive, digitally stored, computer-controlled manipulation of information...

A medium for journalists, artists, designers, musicians, scientists, students, doctors, engineers, ...

Types of media

Underlying technology: **Data compression**

Informal definition of multimedia

Multimedia is the **seamless integration** of text, graphics, sound, images and control software to navigate, interact, create and communicate **within a single information environment**.

Motivation for using multimedia:

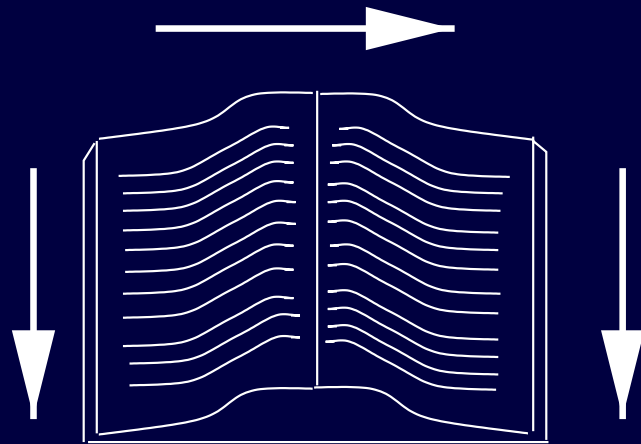
Humans retain:

- 20% of what they **see**
- 30% of what they **hear**
- 50% of what they both **see and hear**
- 80% of what they **see, hear and do**

History: Multimedia developed from computer-based training and interactive video technology.

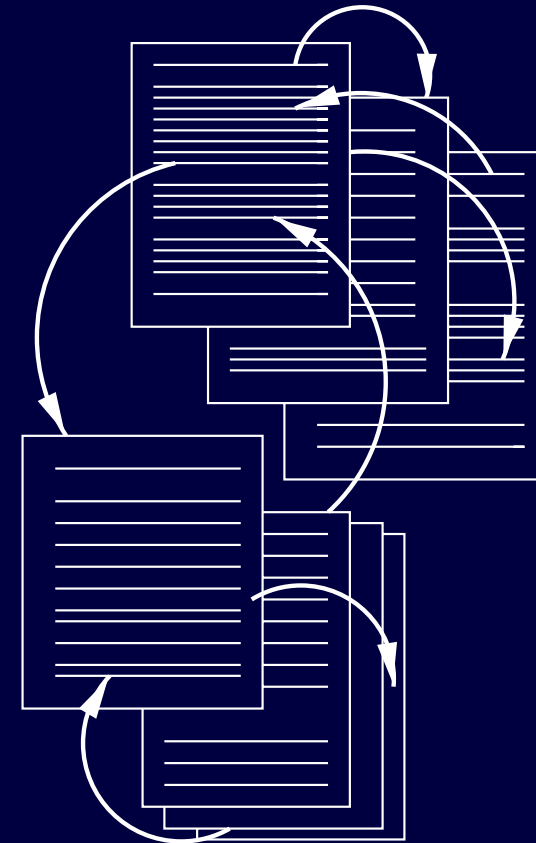
Normal text vs. Hypertext

Normal Text

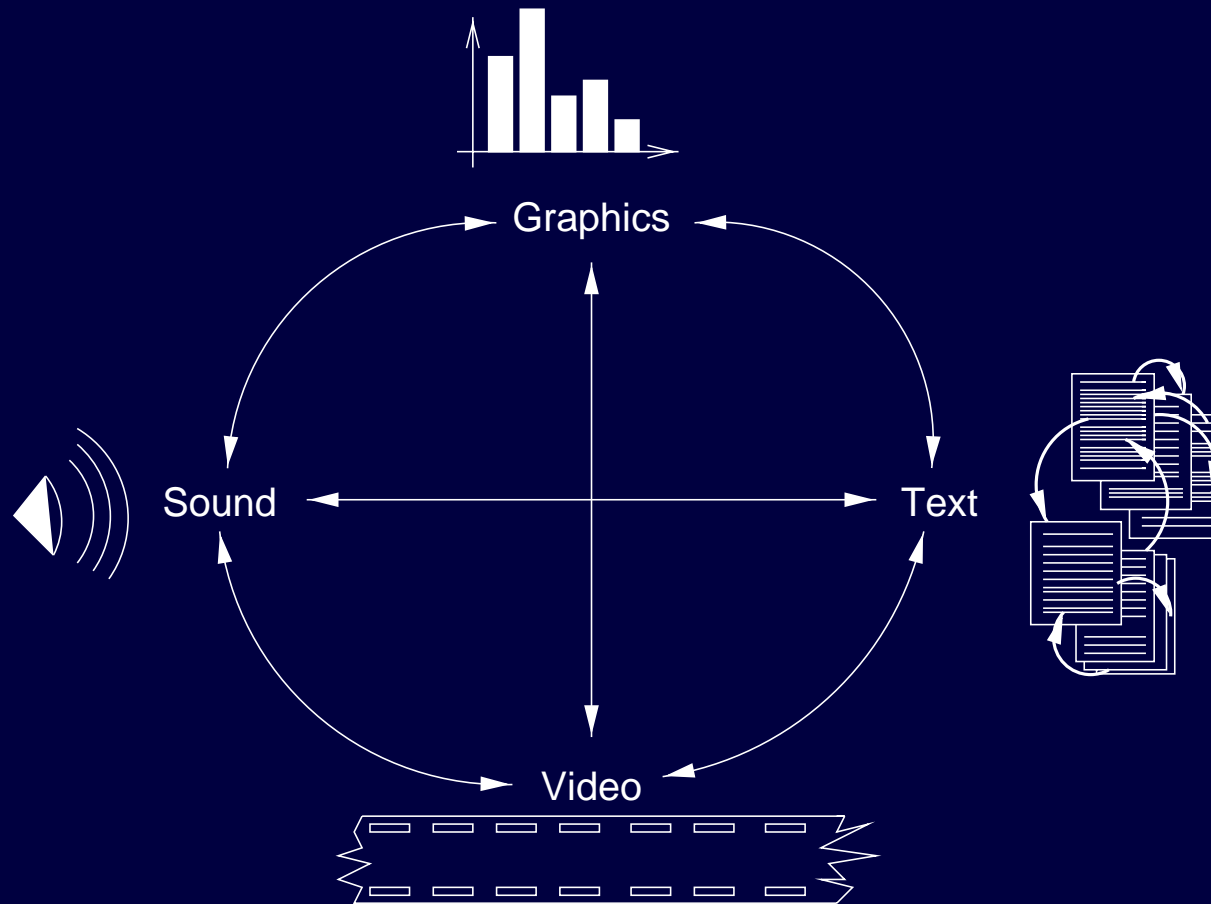


Linear

Hypertext



Hypermedia



Multimedia: **sequential** in terms of information flow.

Hypermedia: **non-linear links** which a user may **follow in any order**.

Content providers view

- Multimedia is about **content**.
 - **Text** (printed, scanned, electronic,...)
 - **Graphics** (bitmaps, clip art, pictures,...)
 - **Sound** (CD audio, musical instrument digital interface MIDI,...)
 - **Video** (live, tape, digitally stored,...)
- Multimedia revolution has been made possible by
 - TV networks (cable TV, uni-directional)
 - Telecommunication networks (bi-directional)
 - Publishers & Content providers
 - Computing power / price

Computer Science view

- Multimedia/Hypermedia are **computing subjects**.
- They simply **use and integrate existing disciplines**:
 - Digital image processing (digital TV)
 - Computer graphics
 - Digital speech and audio
 - Networking
- Have driven content producers to **join forces** with computer and networking companies.
- Constantly push the limits of current technologies.
 - High-definition film-making

Current limitations

Some of the limitations of multimedia are:

- Human-computer interface
- Information overkill
- Speed of use (e.g. WWW)
 - performance of networks \Rightarrow too slow
 - “size” of multimedia data \Rightarrow too large

How do we quantify speed?

Networks exhibit performance characteristics:

- Speed/timeToWait or download
- Latency (single bit propagation time)
- Bandwidth (data **transmitted** per second)

Pipe analogy

- latency related to length/rate of water flow
- bandwidth related to girth/cross-sectional area

We'll return to networks periodically in the course

Kinds of sizes involved

Video compression

One second of uncompressed 32-bit colour, 720x576 resolution, 25fps digital video would require approximately **40MB of storage**.

One minute would require about **2.5GB**

A **CD ROM** can only hold about **700 MB**, and a **1x player** can only transfer **150 KB per second**.

A **DVD-1g** can hold about **4.7 GB** or 117s (of uncompressed video)

⇒ **Data storage and transfer problems**

- Increase proportionally with resolution and frame-rate.
- Limit viability/extent of uncompressed solutions

Data compression techniques (I)

Classification into:

- **Lossless** e.g. 2:1,3:1
 - Decompressed bit-stream is identical to the original.
 - * computer programs, text
- **Lossy** e.g. 10:1,40:1
 - Irreversible, decompressed information is changed from original.
 - (– Why or when is this acceptable?)
 - psychometric testing for algorithm design
 - * video, audio, still images (.jpg files) for human perception

Data compression techniques (II)

Classification into:

- **Statistical encoding**

- Takes no account of the nature of the data.
- Ignores semantics of the information.
- Generally lossless.

- **Source encoding**

- Uses knowledge about the information.
- Higher compression rates may be possible.
- May be either lossy or lossless.

In this introduction course only *statistical lossless techniques*.

Others will be explained in more advanced courses.

Run-length encoding (statistical lossless)

Replace a sequence of n successive values c by

- c itself,
- a special character (flag) and
- the number of occurrences.
 - * audio encoding (pause suppression)
 - * compressing clip art (`.gif` files)

6	8	3	0	0	0	0	0	0	0	0	0	0	0	0	0	7
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

6	8	3	0	F	13	7
---	---	---	---	---	----	---

Huffman (statistical lossless) (I)

ASCII coding of the string “PEPPERMILL” would require a total of $10 \times 7 \text{bit} = 70 \text{ bit}$. Hence **Use a frequency dependent code.**

Develop a Code Tree:

Leaf nodes of the tree are the letters in the string (alphabet).

Arrange nodes in order of probability:

- P: 3/10
- E: 2/10
- L: 2/10
- R: 1/10
- M: 1/10
- I: 1/10

Initially there are as many nodes as different letters in the string.

Sum over probabilities gives 1.

Huffman (statistical lossless) (II)

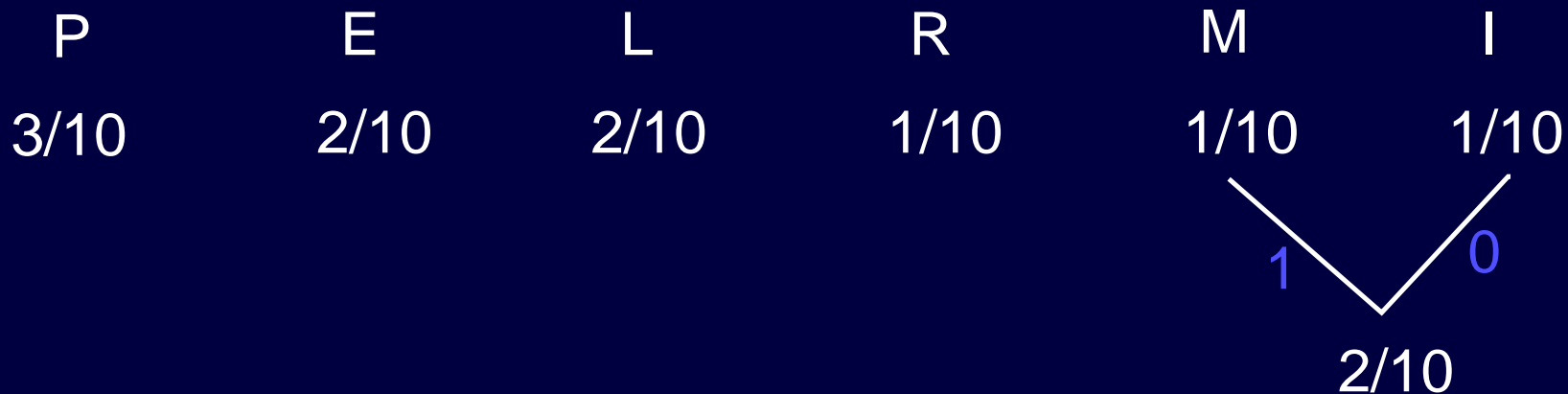
P	E	L	R	M	I
3/10	2/10	2/10	1/10	1/10	1/10

Join two least probable nodes. Form a new node (add probabilities).

Repeat joining nodes, until probability is 1 (root node).

Number right branches with a 0 and left branches with a 1.

Huffman (statistical lossless) (II)

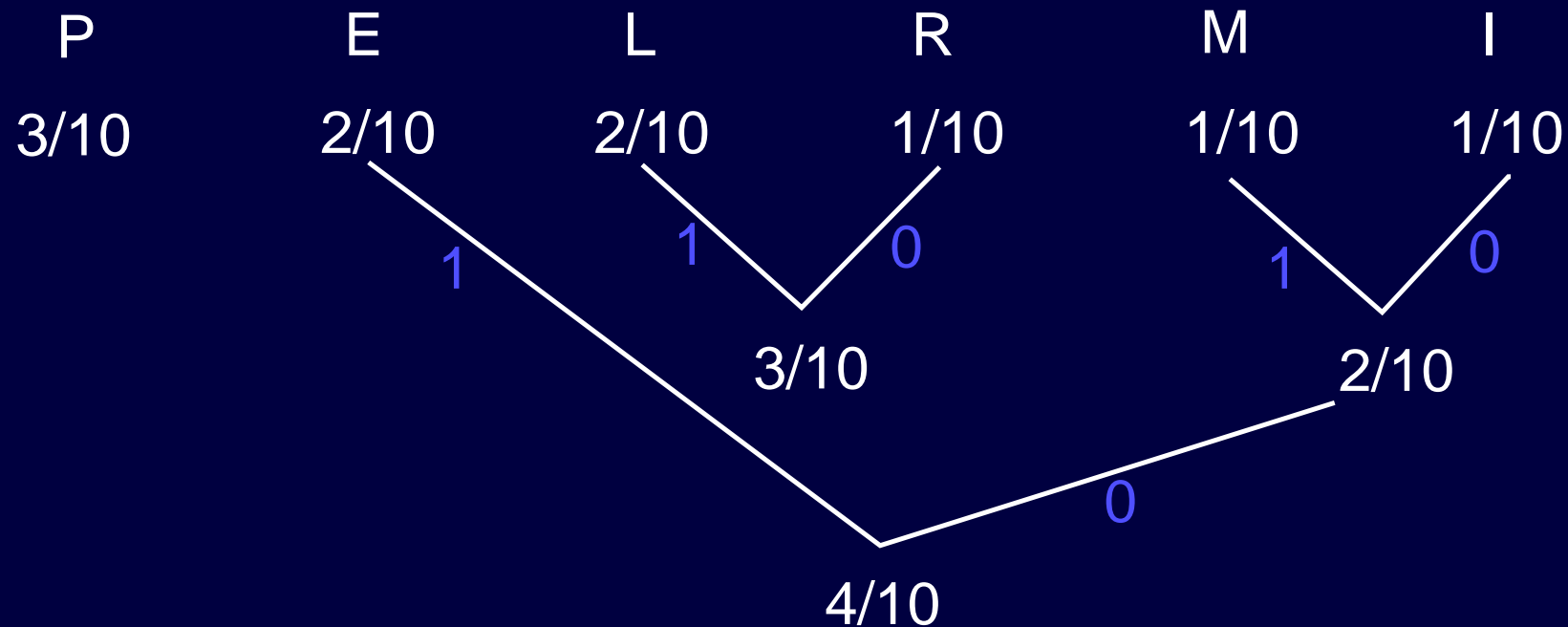


Join two least probable nodes. Form a new node (add probabilities).

Repeat joining nodes, until probability is 1 (root node).

Number right branches with a 0 and left branches with a 1.

Huffman (statistical lossless) (II)

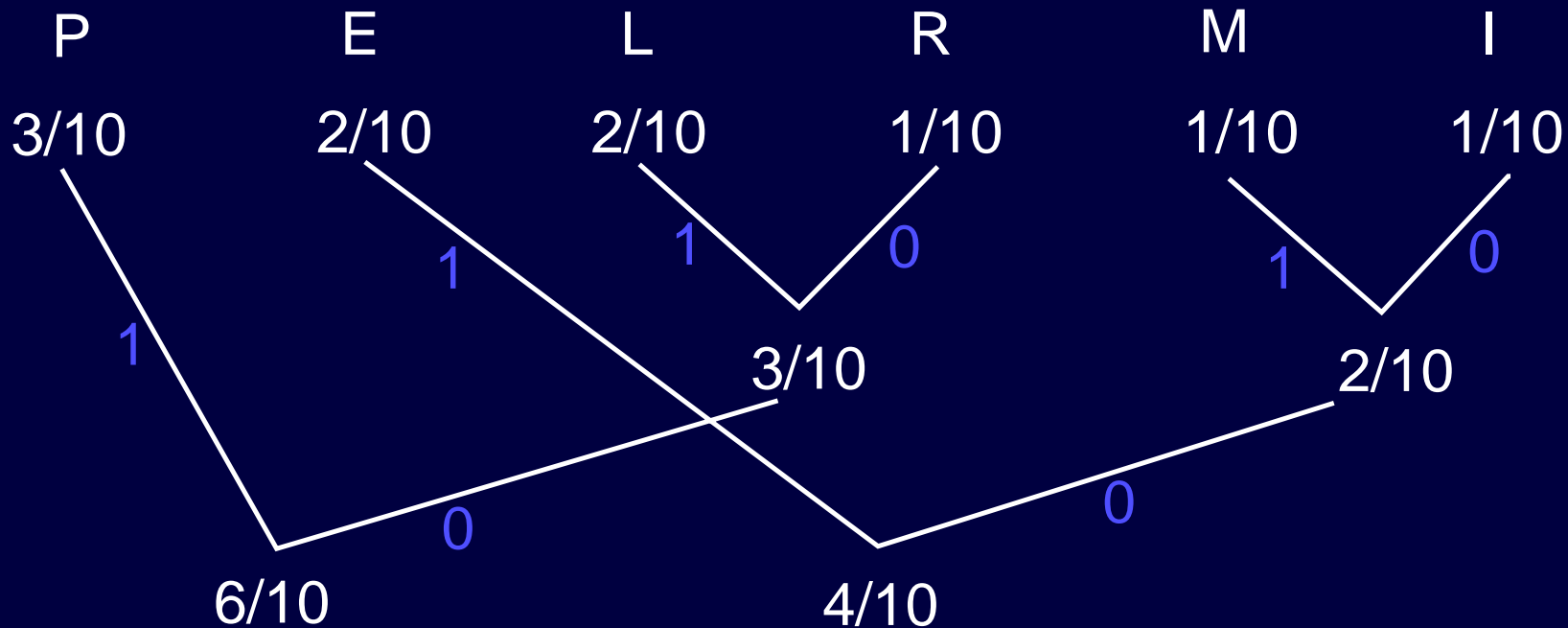


Join two least probable nodes. Form a new node (add probabilities).

Repeat joining nodes, until probability is 1 (root node).

Number right branches with a 0 and left branches with a 1.

Huffman (statistical lossless) (II)

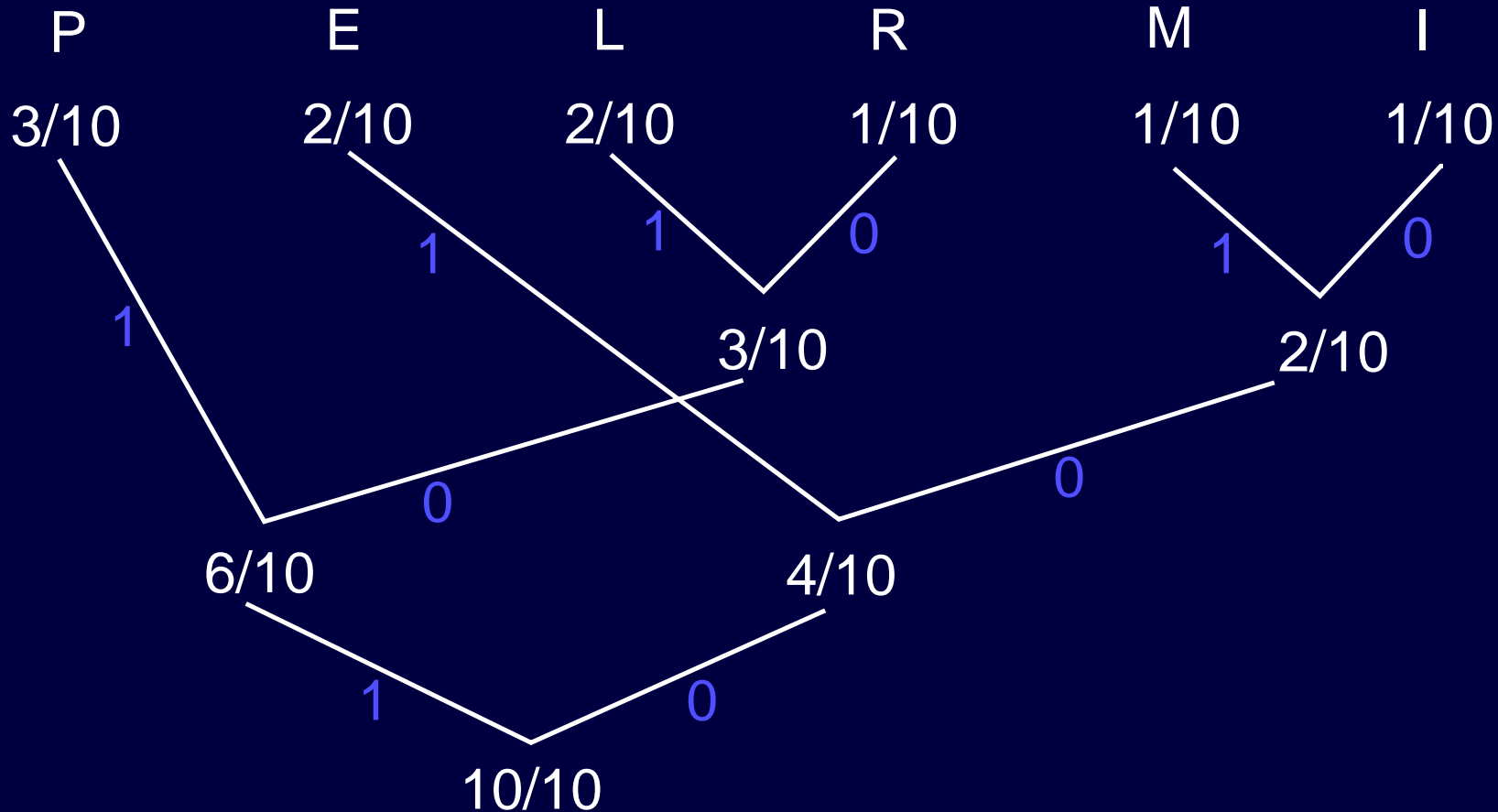


Join two least probable nodes. Form a new node (add probabilities).

Repeat joining nodes, until probability is 1 (root node).

Number right branches with a 0 and left branches with a 1.

Huffman (statistical lossless) (II)



Join two least probable nodes. Form a new node (add probabilities).

Repeat joining nodes, until probability is 1 (root node).

Number right branches with a 0 and left branches with a 1.

Huffman (statistical lossless) (III)

The **code for each letter** can be read from the tree, following the path from the root node to the respective leaf node and collecting the 1s and 0s.

Huffman coding of string “PEPPERMILL” needs 25 bit.

- P: 11
- E: 01
- L: 101
- R: 100
- M: 001
- I: 000

PEPPERMILL is encoded as: **11011111101100001000101101**
00100010110101100 stands for **MILLER**

Properties of Huffman code

- **Frequency dependent** code.
- More frequently used letters (e,t,a,i in English) are represented by **short bit patterns**.
- Less frequently used letters (z,q,x in English) are represented by **longer bit patterns**.

Result: Compression through reducing average code length.

+ **Optimum code**

- Shortest average code length of all statistical encoding techniques.

+ **Prefix property**: No short code group is duplicated as the beginning of a longer group. **Ensures** that code is **uniquely decipherable**.

– **A=0, B=01, C=11, D=00** is NOT a Huffman code

⇒ **0001 = AAB** or **0001 = DB?**

Summary

Today: Investigated Multimedia and compression techniques.

Where to get more information?

- Animation at:

`http://www2.cs.pitt.edu/~kirk/cs1501/animations/Huffman.html`

- Thomas A. Standish “Data Structures, Algorithms & Software Principles in C”

Addison-Wesley, 1995, p. 394 ff (Huffman Coding)

- A. D. Dewdney “The New Turing Omnibus”

Computer Science Press, 1997, p. 345 ff (Huffman Coding)

Concurrency

A *concurrent* system consists of multiple interacting components that are active at the same time.

- Internet Browser: User Interface / Downloading. Interact by means of the rendering engine.
- Airplane: hundreds of components
 - Engine Controller (times 4)
 - Autopilot (times 2/3 if redundant)
 - ...
- Supermarket checkout: Cash Register and Barcode Scanner.

Problems:

- Inconsistencies (Critical Sections, Mutual Exclusion)
- Deadlock and Livelock

What is a concurrent program?

Ordinary program:

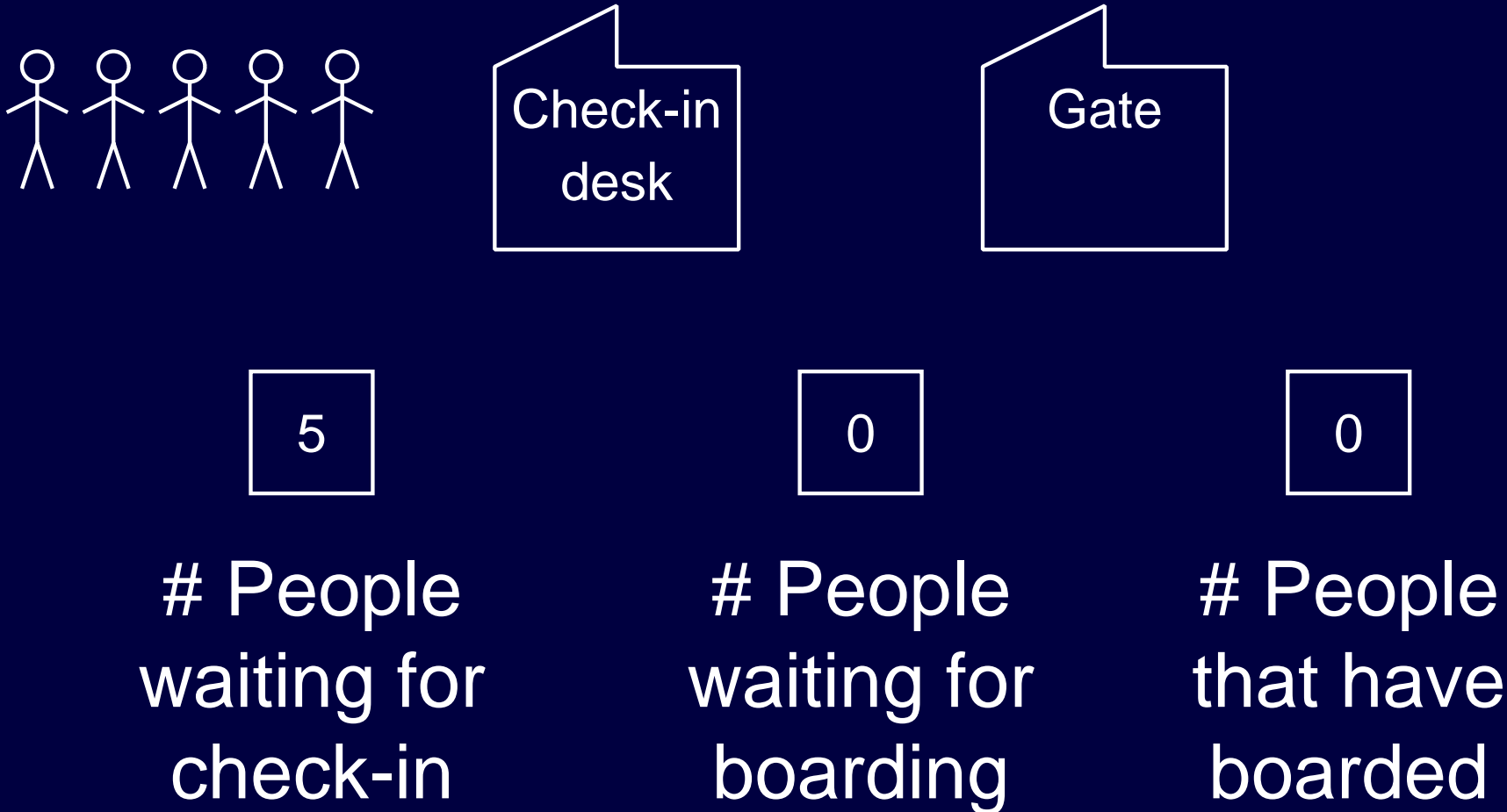
- data declarations plus sequence of instructions
- execute instructions sequentially

Concurrent program:

- set of ordinary sequential programs
 - (• Depending on the programming language/paradigm we call them **process**, or task, or thread, or ...)
 - instructions are executed in **abstract parallelism**
- ⇒ Parallelism is *abstract* because a separate physical processor is not necessary to execute each process.
- ⇒ Hence, we can **analyse the program without knowing on what platform it will be executed.**

Note, could be two programs, or some hardware and software.

Example: Airport Check-in



Example: Airport Check-in



4

People
waiting for
check-in

1

People
waiting for
boarding

0

People
that have
boarded

Example: Airport Check-in



3

People
waiting for
check-in

2

People
waiting for
boarding

0

People
that have
boarded

Example: Airport Check-in



3

People
waiting for
check-in

1

People
waiting for
boarding

1

People
that have
boarded

Example: Airport Check-in



2

People
waiting for
check-in

2

People
waiting for
boarding

1

People
that have
boarded

Program to maintain counter

High-level program, consisting of two processes CHECKIN and BOARD:

INTEGER N = 0; \Rightarrow N is a shared variable!

```
PROCESS CHECKIN() {
    WHILE(1) {
        QUEUE=QUEUE-1;
        N = N+1;
    }
}

PROCESS BOARD() {
    WHILE(1) {
        IF (N != 0 ) {
            N = N-1;
            BOARD=BOARD+1
        }
    }
}
```

These processes are executed in parallel.

- We know that they are executed in machine code, lets take a look at the assembly code...

NEW program to increment the counter

High-level program, consisting of two processes CHECKIN and BOARD:

INTEGER N = 0; \Rightarrow N is a shared variable!

```
PROCESS CHECKIN() {
    WHILE(1) {
        ... Decrement QUEUE
        LOAD  Reg1, N
        ADD   Reg1, 1
        STORE Reg1, N
    }
}

PROCESS BOARD() {
    WHILE(1) {
        IF (N != 0 ) {
            LOAD  Reg1, N
            ADD   Reg1, -1
            STORE Reg1, N
            ... Increment BOARD
        }
    }
}
```

Processes can be interleaved

What is being interleaved?

Depends on instructions of processor.

- Implementation based on **LOAD**, **ADD**, **STORE**.
- Each processor has own set of registers! Here, **Reg1** and **Reg2**.

<i>Process</i>	<i>Instruction</i>	<i>Value of N</i>	<i>Reg1</i>	<i>Reg2</i>
	(initially)	0	-	-
CHECKIN	LOAD Reg1, N	0	0	-
CHECKIN	ADD Reg1, 1	0	1	-
CHECKIN	STORE Reg1, N	1	1	-
BOARD	LOAD Reg2, N	1	1	1
BOARD	ADD Reg2, -1	1	1	0
BOARD	STORE Reg2, N	0	1	0

⇒ **Appears to work fine.**

Finer grained machine instructions

What happens if boarding and check-in happen at the same time?

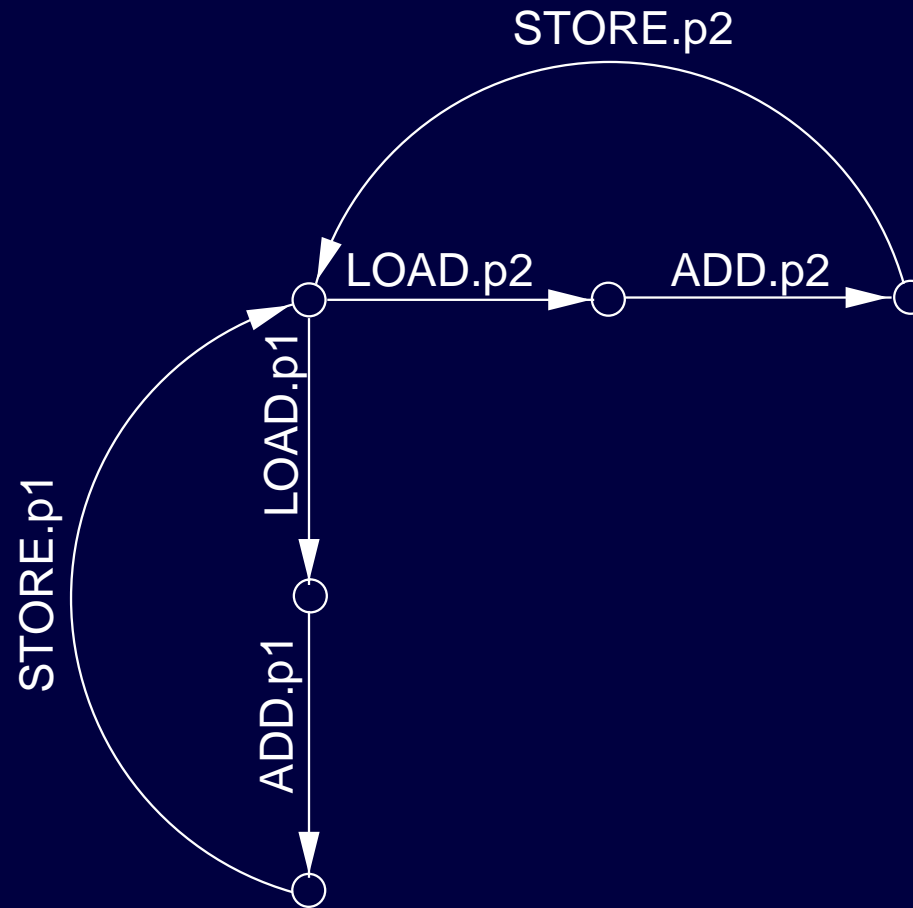
⇒ CHECKIN and BOARD will run *simultaneously*.

Process	Instruction	Value of N	Reg1	Reg2
	(initially)	1	-	-
CHECKIN	LOAD Reg1, N	1	1	-
BOARD	LOAD Reg2, N	1	1	1
CHECKIN	ADD Reg1, 1	1	2	1
BOARD	ADD Reg2, -1	1	2	0
CHECKIN	STORE Reg1, N	2	2	0
BOARD	STORE Reg2, N	0	2	0

💣 There is still one person to board, but the counter is at 0, indicating that there is no one left!

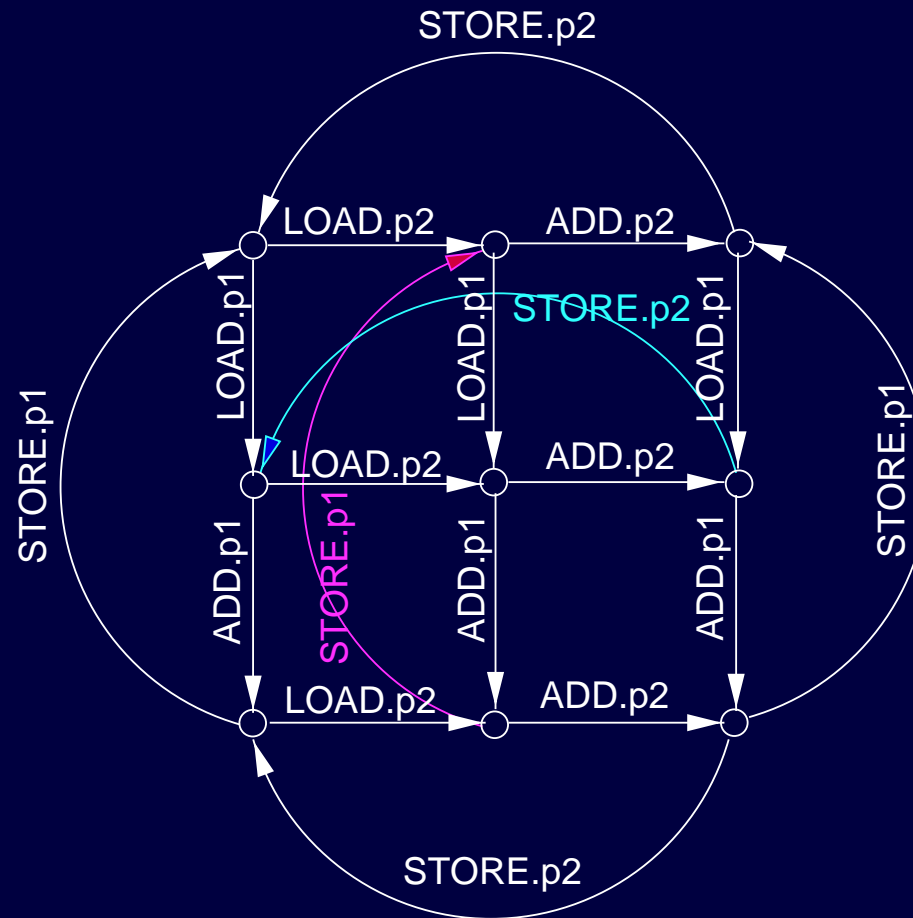
💣 **This interleaving gives the wrong answer!** (Are there others?)

All interleavings of LOAD, ADD, and STORE



These “state spaces” can be analysed, e.g. with process algebras.

All interleavings of LOAD, ADD, and STORE



These “state spaces” can be analysed, e.g. with process algebras.

Critical Sections

Incrementing the counter is a **critical section** of code.

Need to ensure **mutually exclusive access** to the shared variable N .

```
INTEGER N = 0;
PROCESS CHECKIN() {
    WHILE(1) {
        QUEUE=QUEUE-1;
        entry_protocol;
        N = N+1;
        exit_protocol
    }
}
```

```
PROCESS BOARD() {
    WHILE(1) {
        entry_protocol;
        IF (N != 0 ) {
            N = N-1;
            BOARD=BOARD+1
        }
        exit_protocol
    }
}
```

Challenge: Find code for the **entry** and **exit** protocol so that certain **program properties** are guaranteed

Selected Program Properties

Mutual exclusion

- At most one process is executing in its critical section at any time.

Absence of deadlock

- For a deadlock to occur, more than one process tries to enter the critical section but none succeeds.

Eventual entry/Absence of starvation

- A process that is attempting to enter its critical section will eventually succeed.

Absence of unnecessary delay

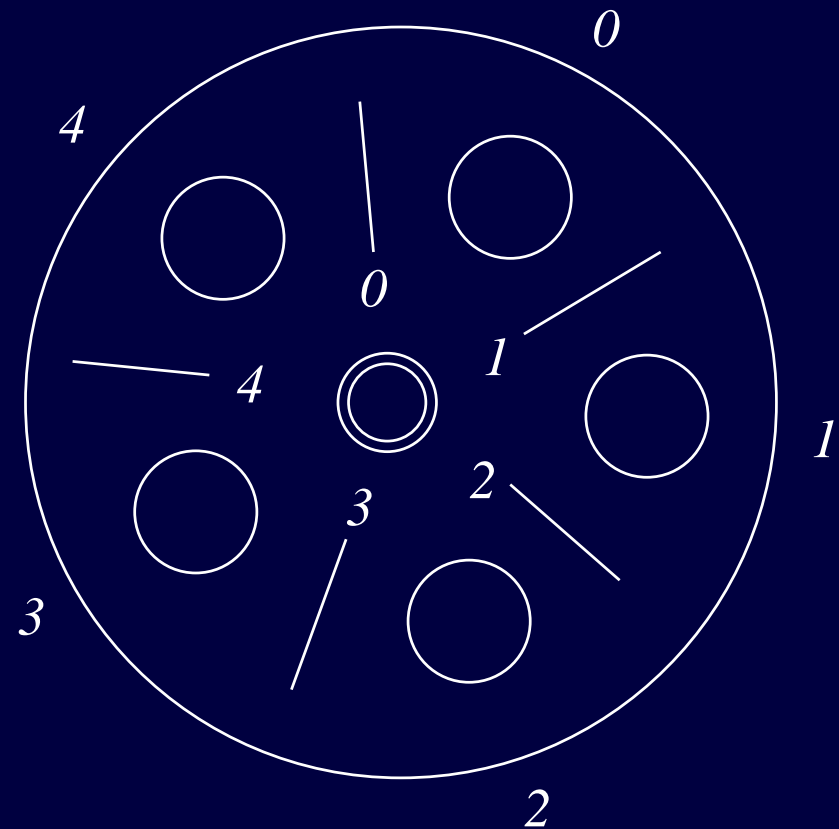
- If a process is trying to enter its critical section and the others are executing outside their critical section or have terminated, the first process is not prevented from entering.

In the next example: Identify the critical section. Where is mutual exclusion required? Can deadlock occur? How about starvation?

Example: Dining Philosophers

Five philosophers live in a college; they spend most of their time thinking, but occasionally become hungry. The college has a communal dining room, with a circular table and five chairs. In the middle of the table is a large bowl of rice, and the table is set with five plates. There are also five chopsticks, one between each pair of plates.

When a philosopher is hungry, she enters the dining room, sits down in her chair, picks up the chopsticks on either side of her plate, and eats. Two chopsticks are needed to eat rice, so if one of the chopsticks is already in use, the philosopher has to wait. When the philosopher has finished eating she puts down the chopsticks, gets up from the chair, and leaves the dining room.



Protecting Critical Sections

- A **critical section** contains statements which access resources that should only be accessed by a restricted number of processes (mostly one) at a time.
 - Critical sections are **encapsulated** by an **entry** and an **exit** protocol.
 - There are different **entry/exit** protocols. (More in COMS22100!)
- = **Basic idea**: Restrict other processes from entering their critical section until it is safe (i.e. the resource is free).
- ⇒ Modern concurrent programming languages provide features for mutual exclusion.

Monitor concept in Java

Concurrent Programming Languages

... are generally based on some paradigm for inter-process communication:

- Message passing
- Shared mutual variables

... need to provide methods:

- for invoking processes (tasks, threads).
- to control these processes.
- to synchronise activities of processes.

Application Areas

- Weather forecasting
- Simulation
- Multimedia
- Distributed database systems
- Graphical user interfaces
- Multi-user applications
- Mobile software
- Games
- ...

Most modern computer applications are based on concurrent and/or distributed programs.

Where to get more information?

M. Ben-Ari

“Principles of Concurrent and Distributed Programming”

Prentice Hall, 1990

J. M. Crichlow

“The Essence of Distributed Systems”

Pearson Education Ltd, 2000

Application areas of AI

Computers can **solve** well-defined **problems** fast and accurately.

⇒ AI aims to **extend** the use of computers to solving less well defined problems, and to deduce, infer and learn. get them to program themselves...

- Game playing.
- Theorem proving (automated reasoning).
- Natural language understanding.
- Computer vision.
- Neural networks and connectionist computing.
- Planning and robotics.
- **Machine learning.**

This lecture: Turing Test, Eliza, Expert Systems and Machine Learning.

A reflection on AI

Sparks imagination, fear, scepticism, ...

Why?

Achievements of AI have not met the initial expectations.

Why?

- Many **problems** turned out to be **more difficult** than originally thought.
 - * Making machines speak and see.
 - * Building machines which learn.
- Original naive optimism \Rightarrow more realistic view on objectives.

Still **enormous potential**.

What is AI? What is I?

Broad term used to **cluster together** areas such as **robotics, game playing, natural language processing**, etc.

Common features:

- To perform activities in these areas, a behaviour is necessary which is considered to be intelligent in humans.

The name AI was given to the entire discipline which investigates how to produce *intelligent* behaviour on machines.

Such machines should display behaviour comparable with what is considered to require intelligence in humans.

The question of intelligence

It is not clear to anyone what intelligence actually is!

- Related to **understanding**.
 - Interpret information in the context of existing knowledge.
 - Assign meaning to information.
- **Knowledge** (objects, attributes, relationships)
 - Possession of knowledge gives a particular **view on the world**.
 - Additions and modifications \Rightarrow Learning

What is so difficult in making computers intelligent?

- Storing knowledge
- Knowledge representation
 - Recognise what is relevant.
 - Knowledge retrieval
 - Inferring and Learning knowledge (expert systems, machine learning)

Can machines think?

What do we mean by **thinking**?

- Rather philosophical issue

⇒ Use **behaviour** as criterion for intelligence.

Some **characteristics** of intelligence:

- planning
- learning
- problem solving
- communicating via a language

Aim of AI:

⇒ Make machines exhibit these characteristics.

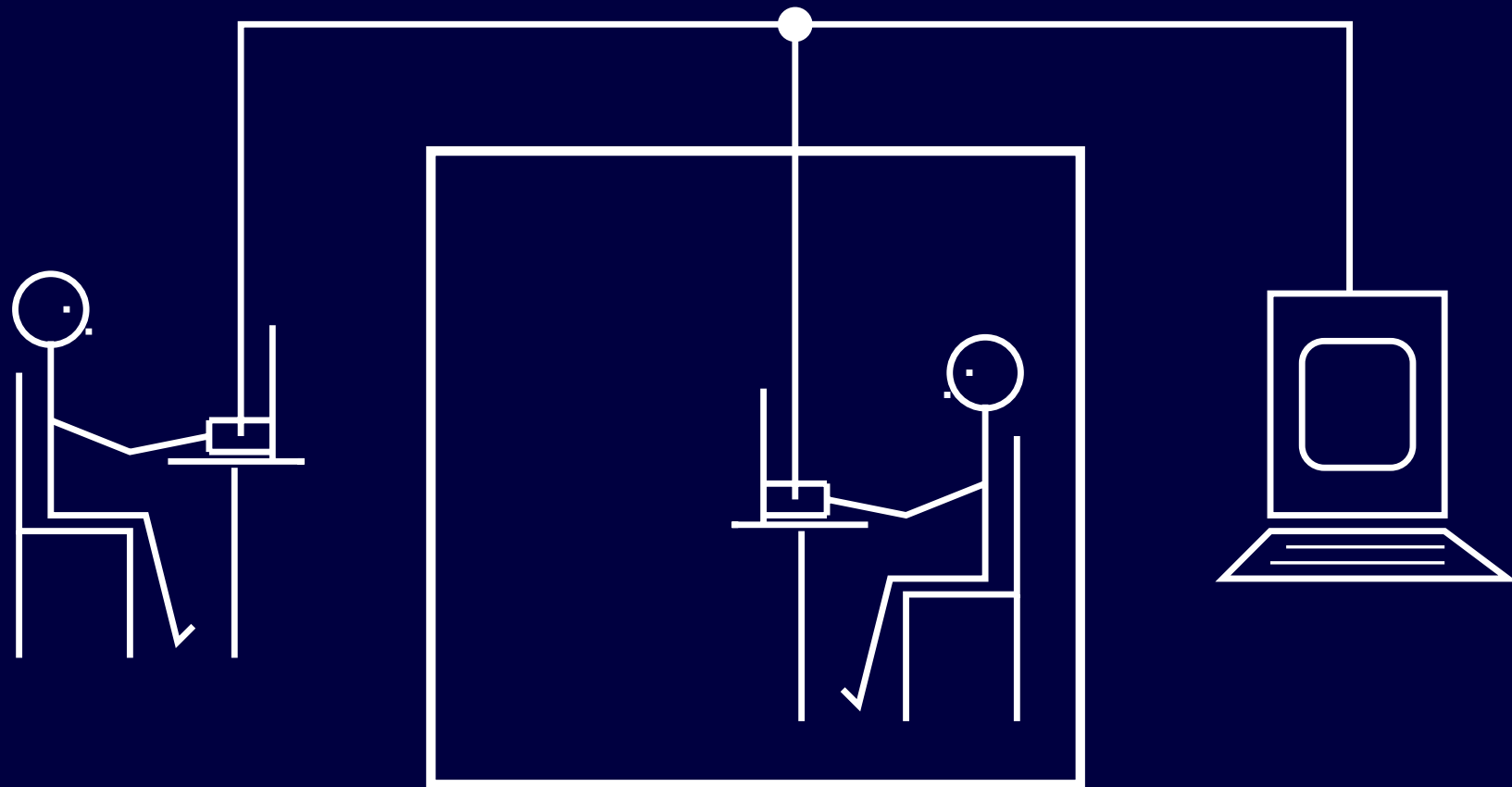
The Turing test (1950)

Aim: Determine whether or not a machine is intelligent.

- 1 human, 1 machine, 1 (human) interrogator
- Interrogator is isolated.
- Only typewritten communication allowed.
- Interrogator must identify human and machine correctly by asking questions.

If the machine tricks the interrogator into believing it is the human, the machine is declared to be intelligent.

A Turing test scenario



Interrogator

Criticism on the Turing test

One (out of many) attempts to define intelligence.

1. Turing test **ignores what happens inside the machine.**
 - Only reflects on observable behaviour.
2. **Biased** towards purely symbolic problem-solving tasks.
 - No test of perceptual skills or manual handling skills.

The Turing test **constraints machine intelligence** to fit the human model.

- *Is machine intelligence different from human intelligence?*
- *Is it right to use human terms to evaluate machine intelligence?*

ELIZA, Joseph Weizenbaum, 60s

Program which emulates responses of a psychiatrist during a consultation with a patient. Example dialogue from Weizenbaum's paper

Patient: **Well, my boyfriend made me come here**

Eliza: YOUR BOYFRIEND MADE YOU COME HERE

Patient: **He says I am depressed most of the time**

Eliza: I'M SORRY TO HEAR YOU ARE DEPRESSED

Patient: **It's true, I am unhappy**

Eliza: DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY?

Patient: **I need some help, that much seems certain**

Eliza: WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP?

...

Analysing ELIZA (I)

Appears to act like a human practitioner.

Satisfies Turing test in a limited domain.

BUT: Responses are echo-like, simply reflecting previous input of the conversation.

⇒ Program based on

- **templates** for sentences
 - * I'M SORRY TO HEAR ...
 - * WHAT WOULD IT MEAN TO YOU IF ...
- set of **keywords/phrases**
 - * "I am", "boyfriend", "unhappy", "help", ...
- set of **transformation rules** for each keyword/phrase
 - * You are XXX ⇒ I am XXX

Analysing ELIZA (II)

⇒ Derive a sentence by

1. Searching for a keyword/phrase in previous input.

* He says **I am depressed** most of the time

2. Select a transformation rule for the keyword/phrase and substitute.

– **I am XXX** ⇒ **You are XXX**

3. Plug result into template sentence.

* I'M SORRY TO HEAR XXX

Analysing ELIZA (II)

⇒ Derive a sentence by

1. Searching for a keyword/phrase in previous input.

* He says **I am depressed** most of the time

2. Select a transformation rule for the keyword/phrase and substitute.

– ; **I am depressed** ⇒ **You are depressed**

3. Plug result into template sentence.

* I'M SORRY TO HEAR **YOU ARE DEPRESSED**

Analysing ELIZA (III)

Human appearance because:

- Remembers earlier parts of the conversation.
- Able to say something if the patient is not constructive enough.
 - * YOU ARE NOT VERY TALKATIVE TODAY
- Design templates and transformation rules to minimise repetition.

BUT: ELIZA does not understand the conversation.

- Patient: I am going to jump out of the window
- Eliza: How long have you been going to jump out of the window?

Expert systems

Goal: Create computer programs that act as expert problem solvers.

An **expert system** is a **knowledge based program** that provides “**expert quality**” solutions to problems in a **highly specific domain**.

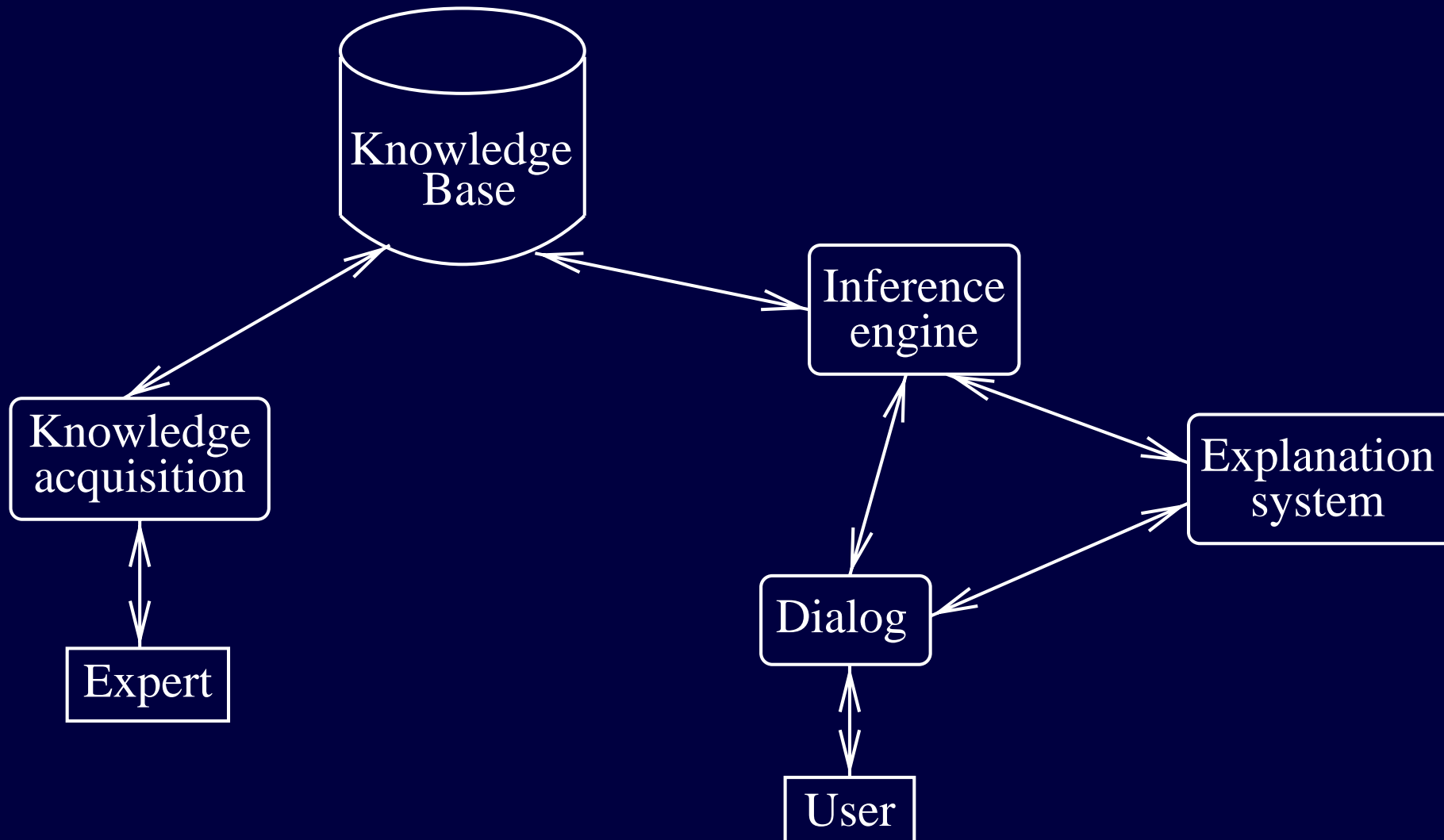
Building an expert system involves:

- **Extract** knowledge from human experts.
 - Ask questions \Rightarrow theoretical knowledge.
 - Observe problem solving strategies \Rightarrow empirical knowledge.
- **Encode** knowledge in formal knowledge representation language.
 - Typically a collection of **rules** and **facts**.

The architecture of expert systems

- **User interface.**
- **Knowledge base (extracted from experts).**
- **Explanation subsystem.**
 - ⇒ Shows “why” and “how” a solution was found.
- **Knowledge base editor (knowledge acquisition).**
- **Inference engine.**
 - ⇒ Means of applying inference rules to derive valid conclusions.

Architecture of a typical expert system



Problems and applicability of expert systems

Problems:

- ; No deep knowledge of domain.
 - * MYCIN (1970s): Pregnancy question for a male patient.
- ; Little (if any) learning from experience.

Applicability:

- + Large body of knowledge available.
- ; Human expertise not available in all situations.
- Problem may be solved using symbolic reasoning techniques.

Machine learning

Goal: Design programs capable of learning.

Learning is the prerequisite for general intelligence.

⇒ A system learns if it **makes changes** in itself that **enable it to better perform** a given task.

Learning strategies

- **Rote learning** = memorisation.
- Learning **from instruction**.
 - Assimilate transformations to knowledge given by an instructor.
- Learning **by deduction**.
 - From general principles to a particular example.
- Learning **by induction**.
 - Find general laws from particular facts, examples, observations and discoveries.
- Learning **by analogy**.
 - ; Check whether it is correctness preserving!

How do machines learn?

Machine learning system components

- **Knowledge base** containing the system's current expertise.
 - * Symbolic, rule-based \Rightarrow Traditional machine learning.
 - * Sub-symbolic, connectionist \Rightarrow Neural networks.
 - * Symbolic, connectionist \Rightarrow Hybrid approaches.
- **Performer**
 - Algorithm that uses the knowledge base to guide its problem-solving activity. It is given a problem and produces a solution.
- **Critic**
 - Feedback module that compares actual with desired solution and measures goodness.
- **Learner**
 - Mechanism that uses feedback from the critic to amend knowledge base.

Where to get more information?

J. Glenn Brookshear

“Computer Science - An Overview” (sixth edition)

Addison-Wesley, 2000

Chapter 10: Artificial Intelligence (p. 437 ff)

Cryptography

The science of hiding information:

- Storing information in such a way that only you can read it (private documents)
- Storing information in such a way that only your colleague can read it (sending an exam paper over e-mail)
- Storing information in such a way that only the bank can read it (digital money)

The flip-side of the coin: Breaking cryptography

- Breaking information stored in secret documents
- Creating digital money.

History

History of cryptography goes to Roman times.

- Caesar Encryption (Julius)
- Every letter is substituted with another letter.
 - Can use modulo arithmetic (26 possible encryptions, the *key* has 26 possible values or less than 5 bits)
 - Can use arbitrary mapping (26! possible encryptions, the key has 403291461126605635584000000 possible combinations).
- Example Modulo substitution:
 - HELLO translates to IFMMP (key is 1) or KHOOR (key is 2)
- Example arbitrary substitution:
 - HELLO translates to ABKKI (key is H to A, E to B, L to K, O to I)

Breaking Caesar I

Modulo substitution can be broken by *brute force*

- Brute force means you try every single key, and see when the message is readable
- Feasible with short keys (only 26 values to try).
- Say you can try one key every microsecond, then you can try 3×10^{13} keys in a year.
 - ⇒ Brute force becomes difficult when you have 10^{16} key values.
(3 centuries *with current state-of-the-art*)
- That is why modern systems have big keys (10^{300} digits)

Breaking Caesar II

Breaking the arbitrary mapping is impossible using brute force

- Observe HELLO is mapped to ABKKI. Every L is mapped on a K.
- Hence, count a letters in a cryptographic message.
- The one that appears most frequent will be the E.
- The one that appears next frequent will be the T.
- Any letter that appears frequently after T might be H.

This uses statistical properties of the language encoded.

- Works if you have prior knowledge of the encoded message.
- Statistics is a major tool in cracking cryptographic systems

History – Enigma (WW II)

Enigma was a big step forward in crypto systems.

- Enigma used a substitution, but a different one for every character.
- Mechanical device, changed the rotors on every character.
- Initial setting of the rotors is determined by the key.

Weaknesses

1. Enigma never mapped a letter onto itself
2. Enigma settings had to be passed
3. Lazy operators used to lean on the 'L' key for a while, or use the code word 'BERLIN' or 'HITLER'

If everything else fail lay mines in a minor port.

- Intercept the message of a panicked operator
- You will be sure to find the word 'MINE' in there...

History – Enigma II

Enigma was broken by Alan Turing, and a large group of people.

- Breaking Enigma was instrumental in designing the first computers.
- Breaking Enigma was instrumental in inventing the concept of an algorithm, as a series of repeated computations that you could program in a computer.

Turing had a large group of people operating like a computer.

- Turing would set them tasks to solve.
- Many would be unaware what they were doing!

In the end, the first computer was developed to do a partial brute force attack.

⇒ Colossus. 1500 valves.

- Paper tape would be read at 30 mph!
- Did the trick :)

What are the essential issues in a crypto system:

- Redundancy in the source text
 - Taking redundancy out (Huffman code!) Prevents statistical attacks.
- Symmetry?
 - Are you going to decode it yourself, or is it something that is intended for another person (and one other person only)?
 - Is anyone going to decode it?
- Do you want to be able to perform operations on Cryptographic data.
 - Paying money from A to B
 - Digital cash.

Redundancy – Entropy

Squeeze the entropy out by compressing the data first.

- For example, by using a Huffman code.

You can also make a digest

- Use a hash function to map your string onto a fixed number of bits.
A signature.
- The number of bits is large enough so that the odds are that no two messages map on the same digest.
- Make a digest of a public contract:
 - Contract cannot be changed without changing the digest.
- Cryptographic hash algorithms are *one-way functions*.
- Computationally it is not feasible that two messages hash to the same checksum.

⇒ Message Digest version 5 (MD5): Most widely used cryptographic checksum algorithm.

Symmetry - I

Passwords under UNIX are encrypted.

- Eight character password encrypted to 12 character code.
- One way function only, ie, there is no *known* way to go back.
- When you type your password, the password is encrypted and then compared to the stored encrypted code.
- Your password is not stored anywhere in plain text!

Symmetry – II

Private and public key encryption

- Public key encryption relies on the fact that there are two keys.
- My private keys allows me to read encrypted documents.
- My public key allows other people to encrypt documents destined for me.
- Alice can encrypt a document that she wants to send to Bob using Bob's public key.
- Bob can decrypt it thanks to his private key.
- If Alice wants, she can prove that it was her who send the document by first encrypting it with her private key.
- Bob decrypts it using his private key, and Alice's public key.

Where do you find crypto

- In your mobile phone.
 - Smart card has a secret in it, it means that Orange can send you the bill for your calls.
- In credit cards
 - There is a secret in there which means that people cannot copy credit cards.
- In set top boxes.
 - Smart card so that you can pay for what you see.
- In ATM machines on the street.
 - So that your account is debited.
- In Netscape
 - So that no one can intercept your credit card details.

Where to get more information?

J. Glenn Brookshear

“Computer Science - An Overview” (sixth edition)

Addison-Wesley, 2000

3.5 Networks (p. 141 ff)

Peterson and Davie

“Computer Networks: A Systems Approach”

Morgan Kaufmann, 1996