2

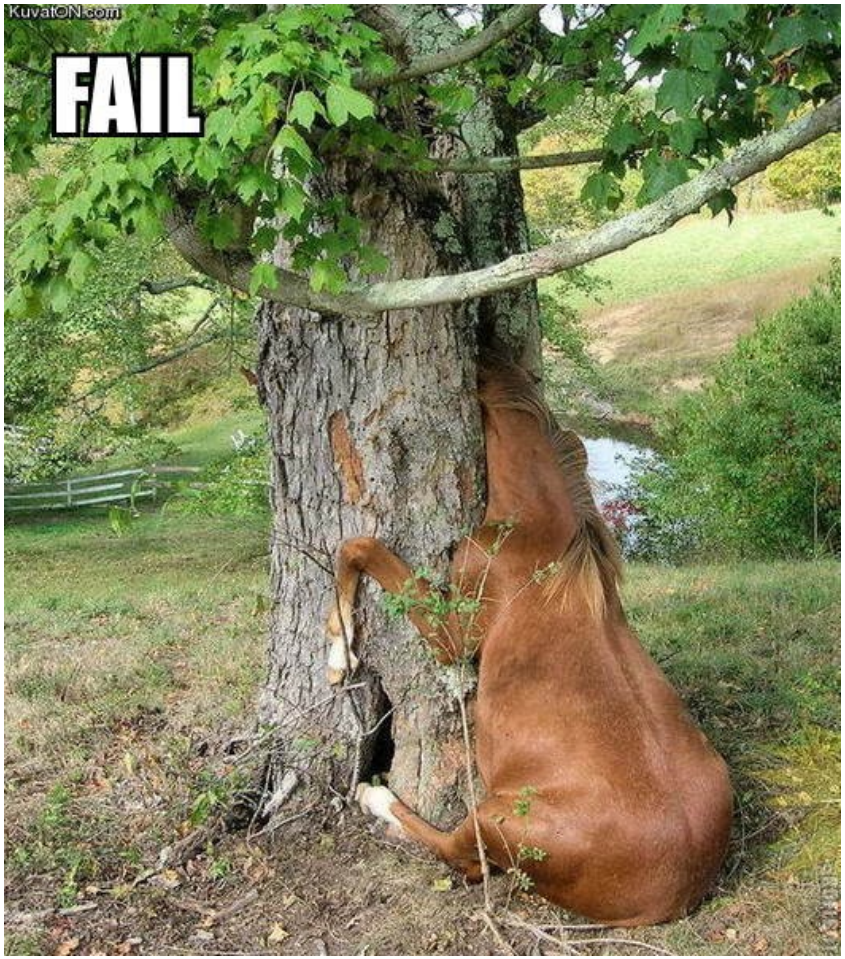
- The correctness problem
  - the machine and the world
  - users are part of the world
  - users & customers
  - distribution infrastructure

The correctness problem

- When is a software system **correct**?
  - "it compiles fine!"
  - "it works!"
  - "it has a nice GUI!"
  - "it is **k**▪**o**▪**o**▪**l**!"
  - "it does what it advertises"
  - "it was delivered in-time"
  - "the letter of the contract was respected"

The correctness problem

- When is a software system **correct**?
  - the implementation fully respects the specification
  - passed all the test suites successfully
  - we are getting 0 support calls on the hotline
  - we had our best team on the project
  - customer did not sue us



The correctness problem

- An interesting product: the DVD rewinder

The correctness problem

- An interesting product: the DVD rewinder

The correctness problem

- **A software system is correct if, once deployed, it brings the world into a desired state**
- Essentially, correctness = satisfaction of requirements
- Provided that
  - requirements are stated w.r.t. the state of the world, not w.r.t. the system itself
  - requirements are feasible w.r.t. available resources and external constraints

The correctness problem

- For the formally-inclined:

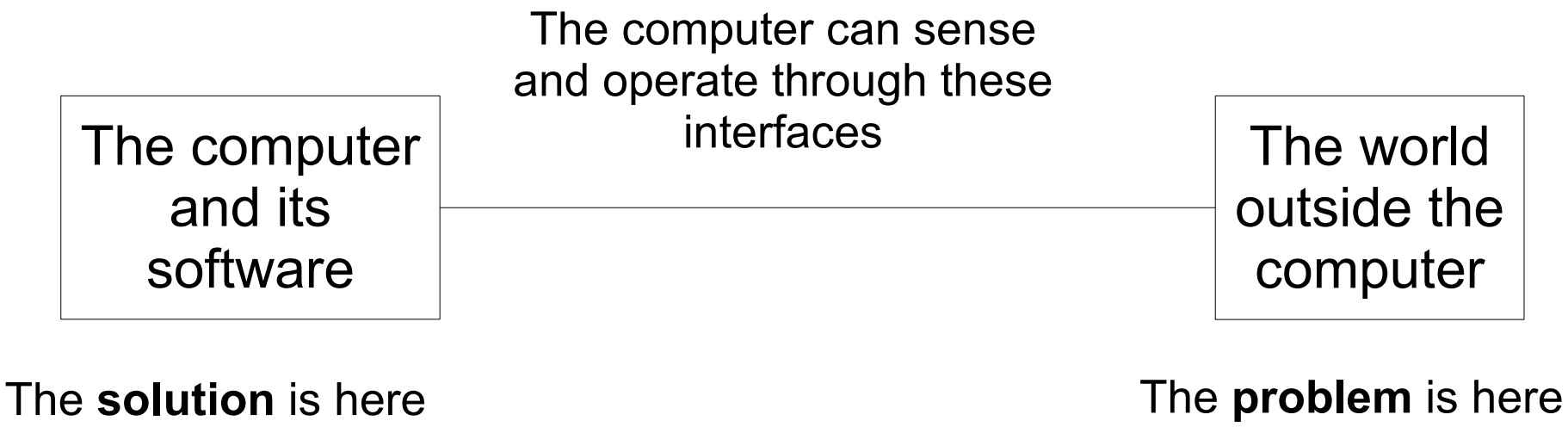
$$S \cup D \models R$$

- That is: the behaviour of the system, as specified by *S*, deployed in the world, as described by *D*, entails (i.e., satisfies) the requirements, as documented by *R*

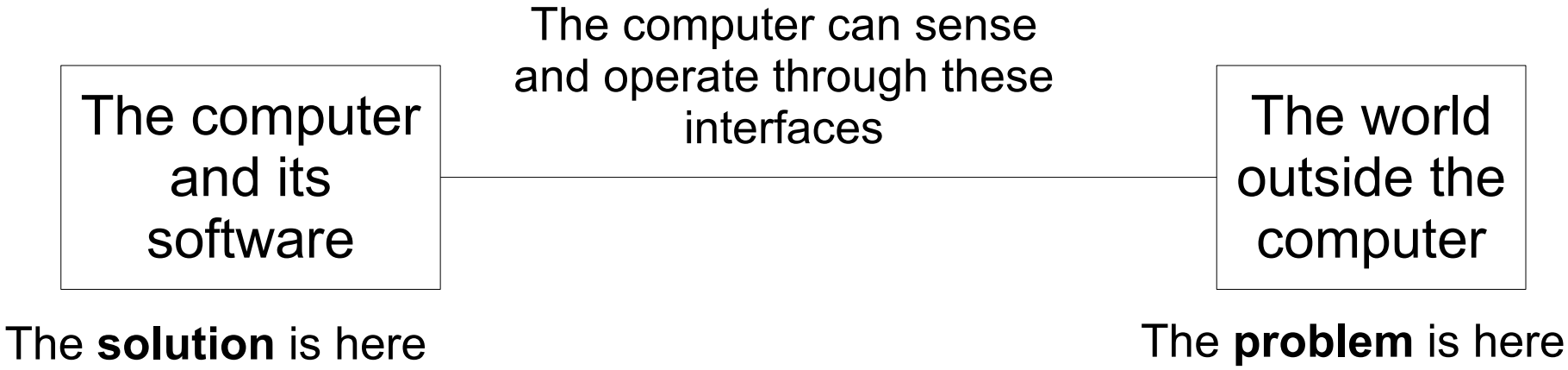
The machine and the world

- Following Michael Jackson's (not the singer) terminology, we will distinguish three elements
  - the **machine**
    - typically, computer+software (the software turns an all-purpose computer into a specialized machine)
  - the **world**
    - everything, but the machine
  - the **interfaces**
    - shared phenomena between machine and world

The machine and the world



The machine and the world



- The problem is **not** in the machine
  - "The system shall do this and that" - wrong!
- The problem is **not** at the interface
  - neither GUI nor sensors/actuators
- Problem is farther away from the computer
  - e.g., what do people do while they are not interacting with the computer?

Structuring the world

- “The world” is a short word for something so large
- In practice, we structure it in a number of **problem domains** which help in bounding the problem
- Each domain abstracts a cohesive part of the real world (from the point of view of our problem)
- Problem domains and machine domain interact through **shared phenomena**

Structuring the world

- Users are modeled as a domain
- Human-computer interaction is described through phenomena shared between users and machine
- Not only GUIs

Structuring the world

- Communication infrastructure can be modeled as a domain
- Many relevant phenomena!
  - Delays; Faults & Failures; Cost; ...

Structuring the world

- Some distribution infrastructures are part of D
  - Internet
  - Cellular networks
  - Land lines connecting shops to banks
  - Credit card marketing and distribution
- Others can be designed and are thus part of S
  - The high-speed communication backplane in a massively parallel computer
  - The exact format of which SMS are sent by a car theft tracking device

An example

- The Patient Monitoring Problem
  - (from M. Jackson, “Problem Frames”, 2001)

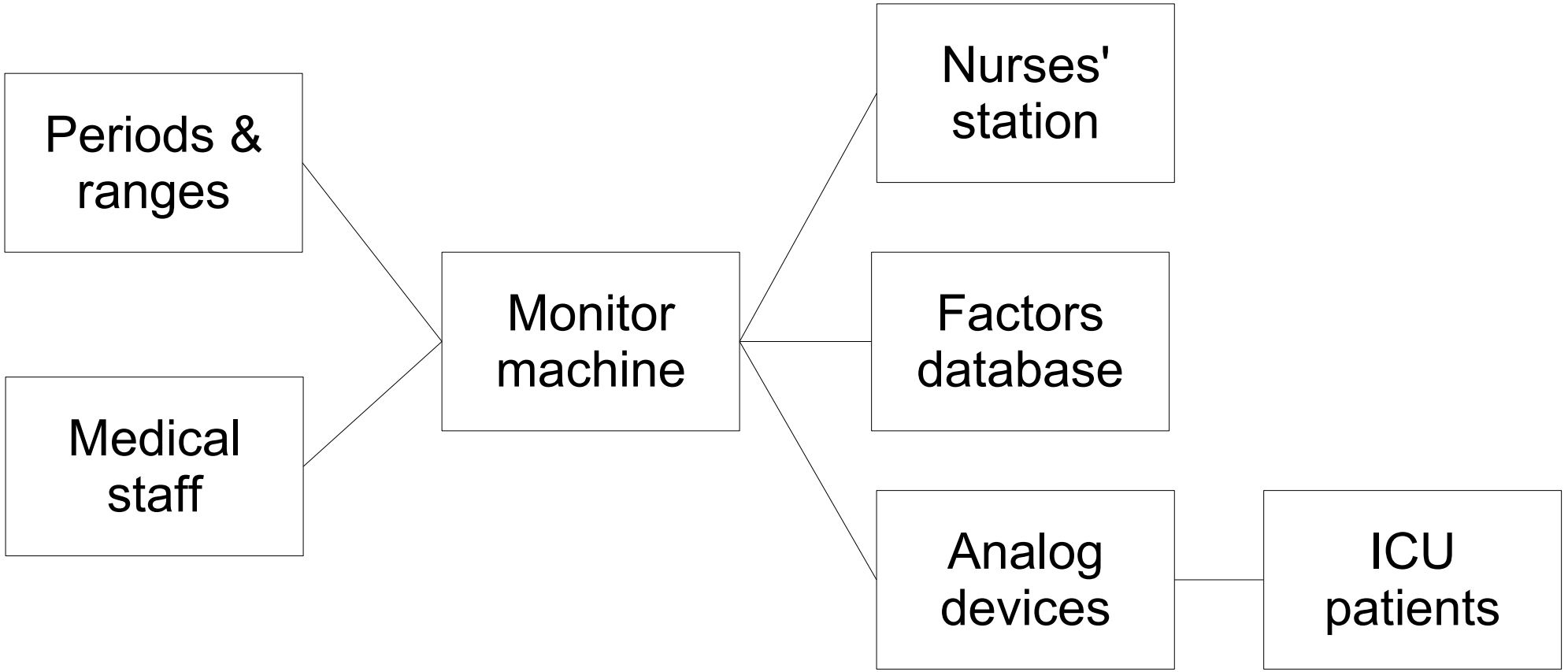
A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure, and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

An example

- The Patient Monitoring Problem
  - (from M. Jackson, “Problem Frames”, 2001)

A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure, and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

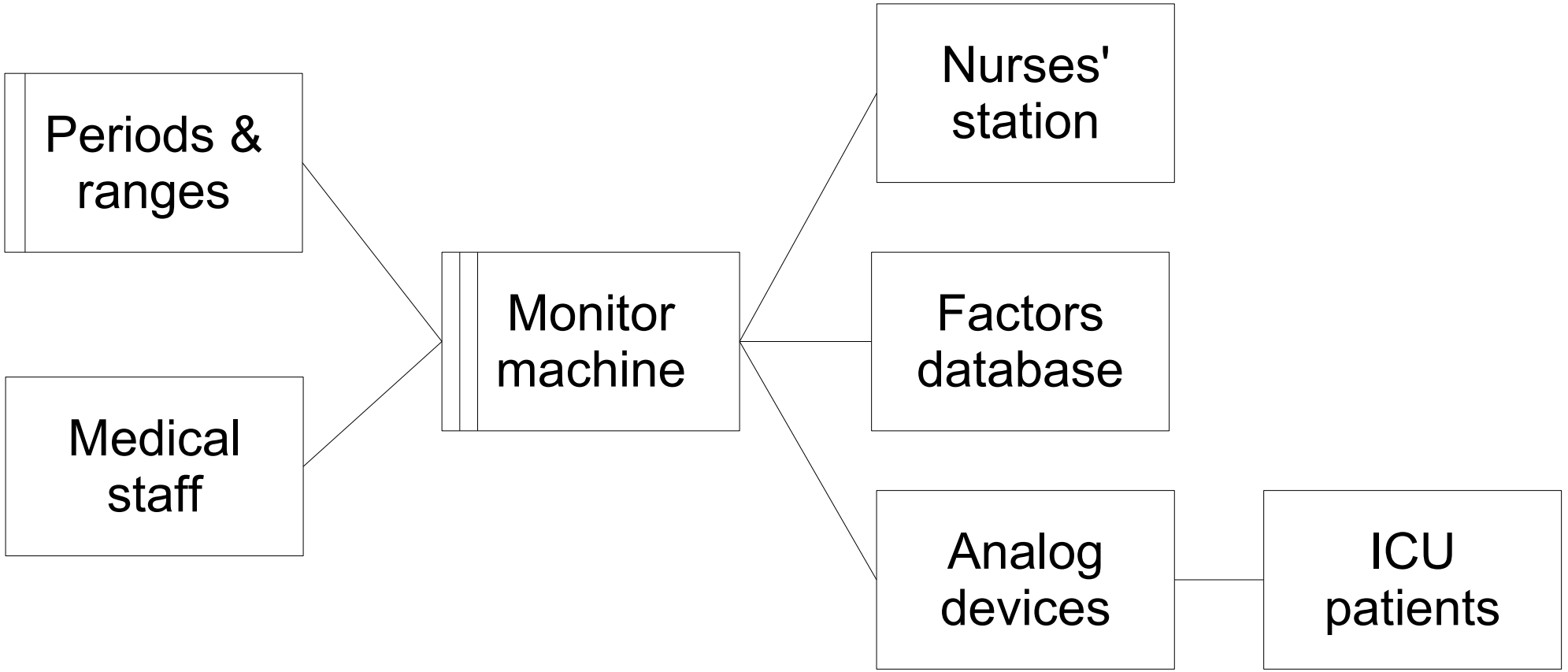
An example


Periods & ranges
Medical staff
Monitor machine
Nurses' station
Factors database
Analog devices
ICU patients

Domains

- All domains are **physical**: they must be **observable** in the real world
- Three types of domains
  - **machine** domain
    - the computer and its software; to be designed
  - **designed** domain
    - a physical representation of some information (e.g., screens, data on disk, printouts), can be designed (at least in part, e.g. layout)
  - **given** domain
    - all properties given, to take “as is”

An example

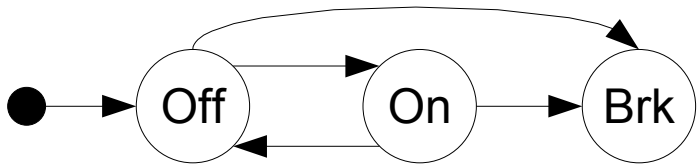

Periods & ranges
Medical staff
Monitor machine
Nurses' station
Factors database
Analog devices
ICU patients

Domains

- Moreover, domains can be:

C

  - **Causal**, if they guarantee certain causal relationships so that we can predict, given some phenomenon, how the domain will react
    - example: a light bulb is a causal domain; given suitable current, it will emit light
    - counter-example: it could break!
    - counter-counter-example:



Domains

- Moreover, domains can be:
  - **Biddable**, if they can be asked to do something, but we have no guarantee that they will react accordingly
    - most often, a biddable domain will consist of people
    - humans don't like being given instructions by a computer, yet in a whole-system view this is very common
    - example: bill of shipment to be loaded on a truck

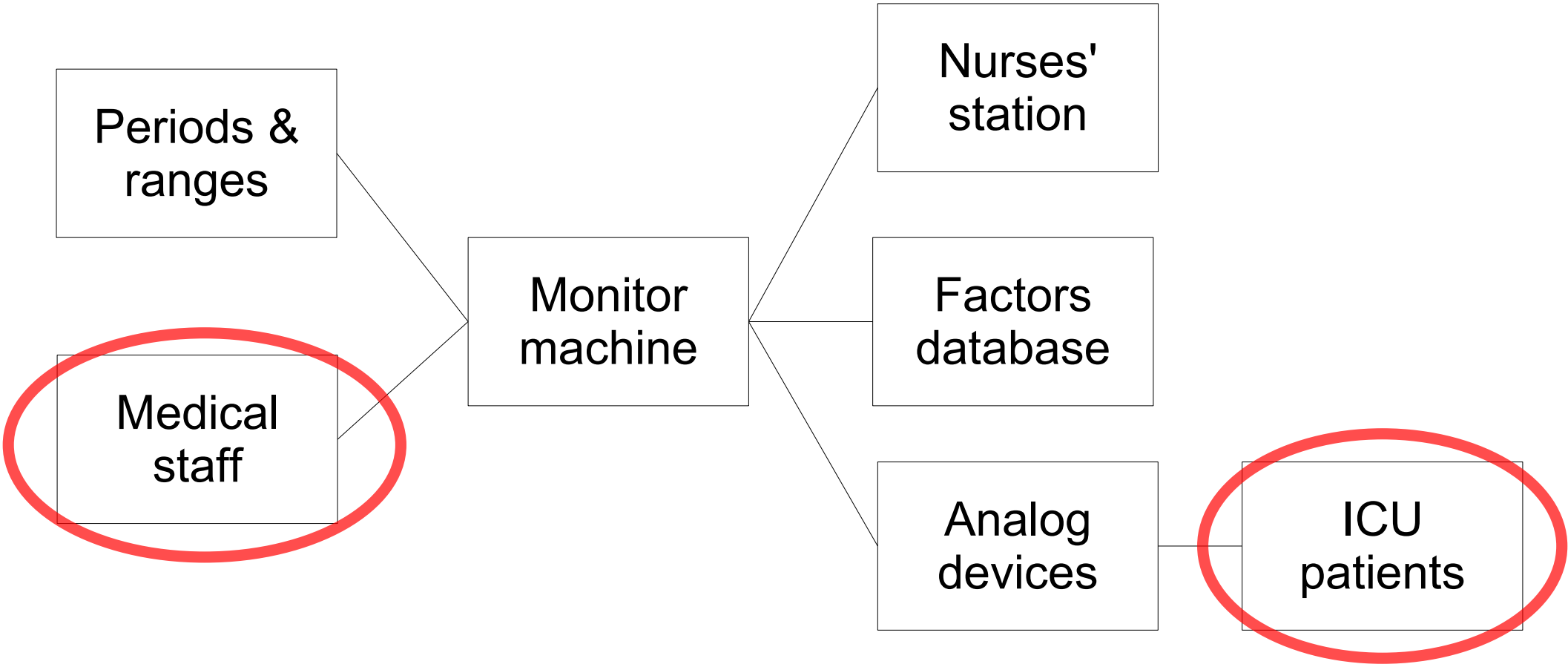
Domains

- Moreover, domains can be:
  - **Lexical**, if they consists of symbols to be manipulated, or in general of a physical representation of data
    - lexical domains share some trait of causal domains, e.g. write and read of data will (usually) work as expected
    - in most other respects, they can be treated as purely passive data
    - example: source program for a compiler

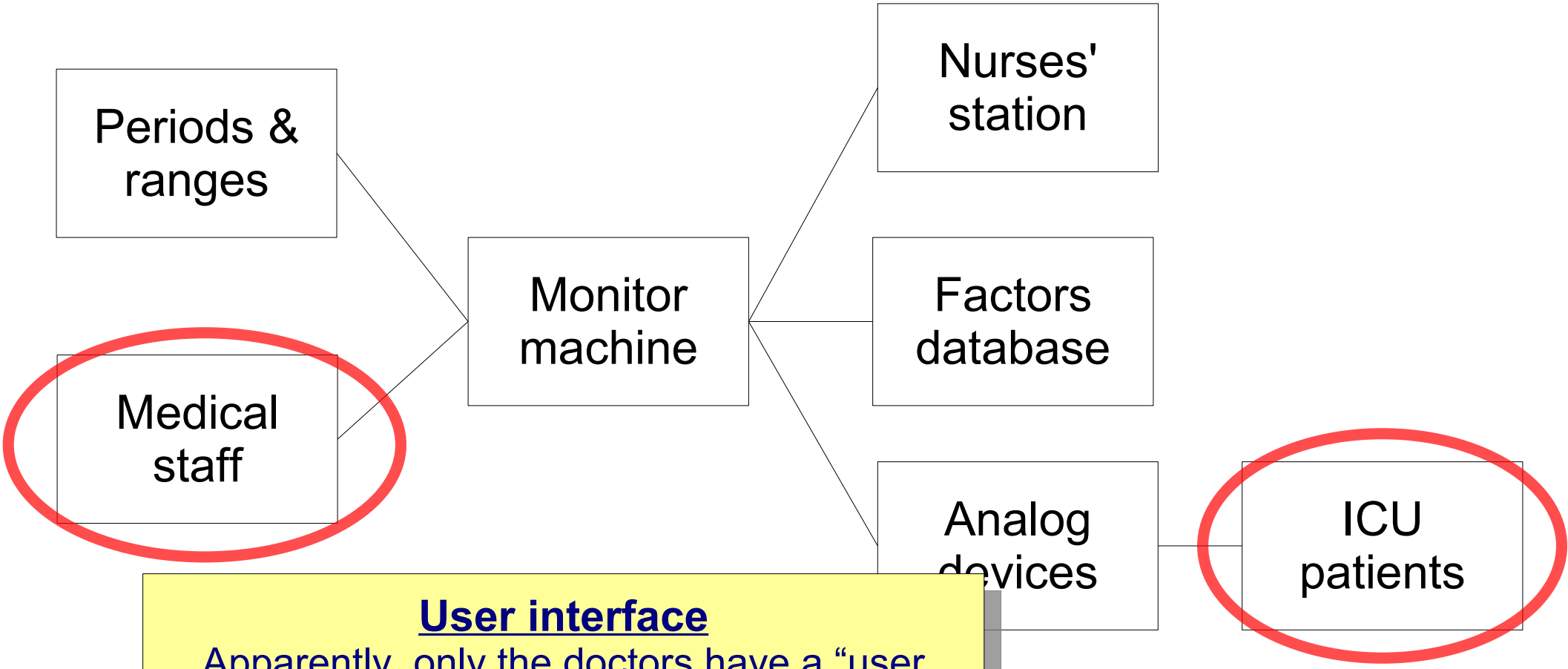
Users are part of the world

- Users are typically a *biddable given* domain
  - the machine can only ask and hope for the best
  - the designer cannot “program” the users
- A large part of the analysis focuses on the phenomena shared between the machine and these domains
  - the right phenomena: correctness
  - suitable physical counterparts: interface design
  - efficient and robust

Humans: not only users



Humans: not only users

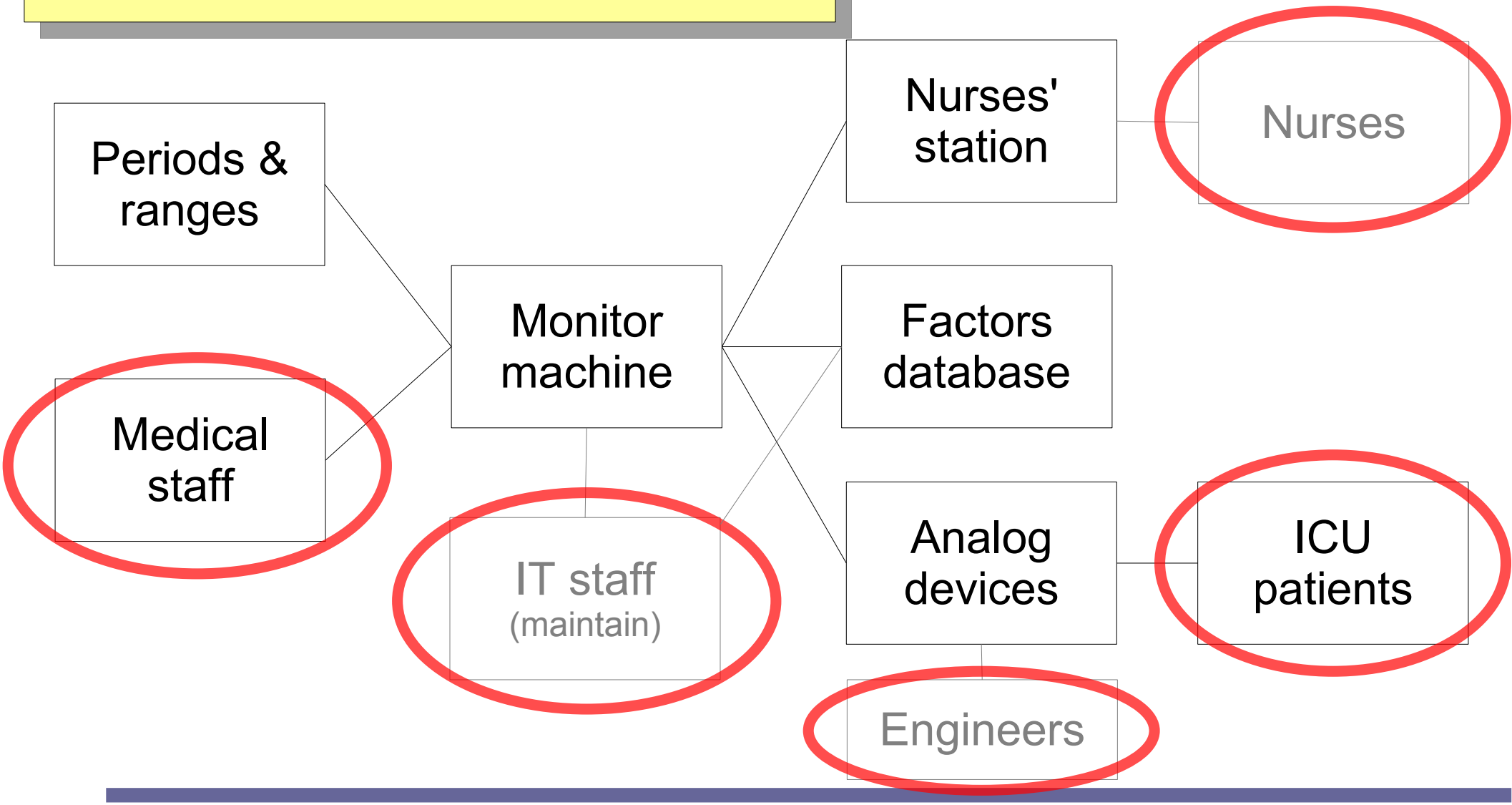


User interface

Apparently, only the doctors have a “user interface” to the machine (to enter a patient's periods&ranges)

sers

User interface
But many other humans have a role in the case: the overarching goal is to provide for the health of the patients



Users and customers

- Typically, the customer pays for the satisfaction of users' desires
- But that is not always the case:
  - more efficient ticket-machines on buses
  - border surveillance applications
  - peer-2-peer blocking software
  - digital rights management
  - locked-down systems
- (there is always the “higher good” excuse)
- The customer pays, his voice is higher!

Distribution infrastructure

- "Best-effort" communication schemes are **biddable**
  - You can ask deliver, hope it will happen
  - Example: UDP over IP
- "Guaranteed delivery" communication schemes are **also** biddable!
  - Wait! Wasn't TCP/IP supposed to be causal?
  - Yeah... 404 NOT FOUND
- It is rarely safe to assume **causal** infrastructure
  - Very robust infrastructure, or
  - Low-assurance system

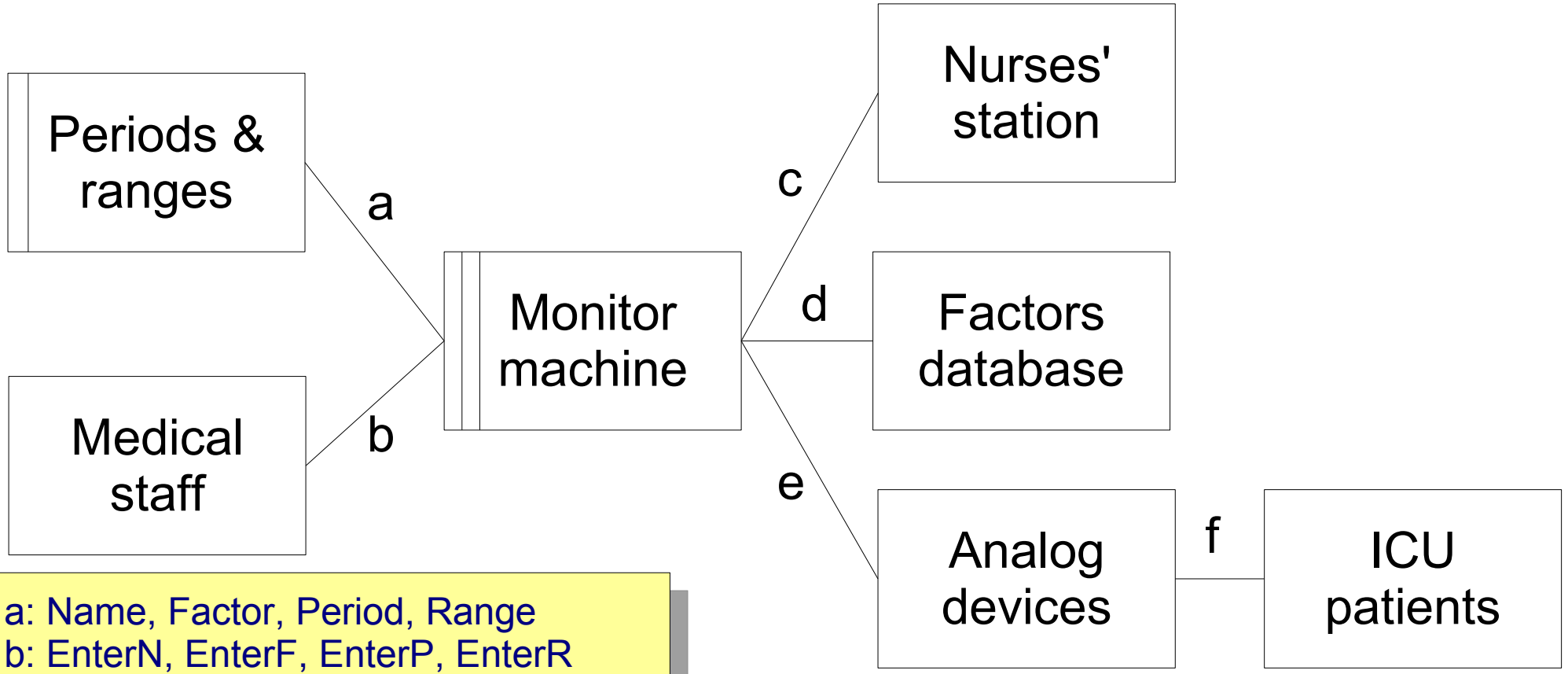
Interfaces as shared phenomena

- It is important to distinguish the notion of **domain interface** from that of user interface
- A domain interface is **a set of shared phenomena** (events, states, values, ...)
- All domains in an interface **participate** of the phenomena; only one can **cause** a given phenomenon
- Phenomena can carry information (parameters)

Types of phenomena

- Individuals
  - Events: an instance of something happening
  - Entities: something that persists over time
  - Values: an element from same (algebraic) sort
- Relations
  - States: relations between entities and values
  - Truths: predicates between individuals (static)
  - Roles: relations between events and individuals
- We will skip the formal parts and rely on intuition in the following

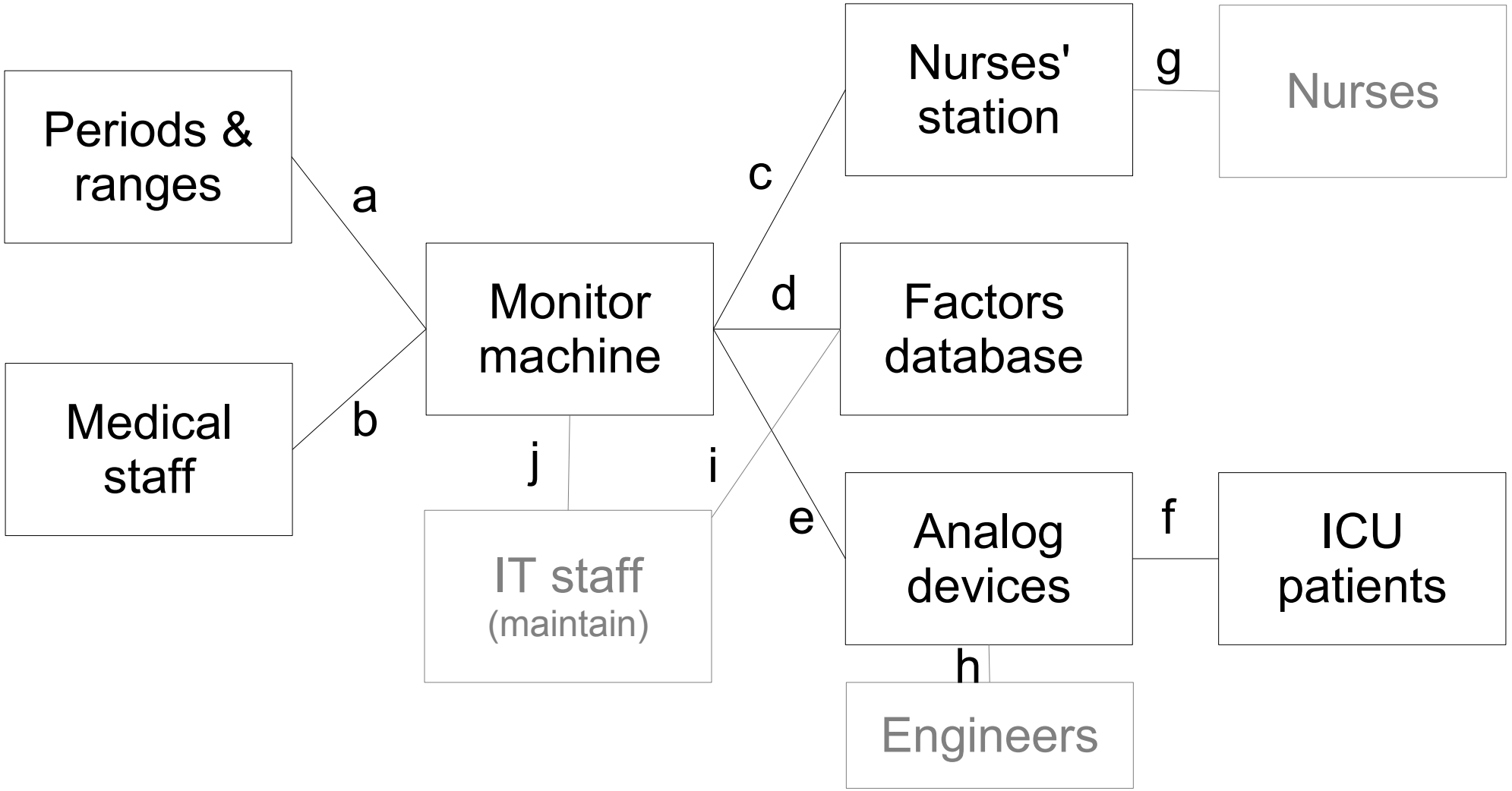
An example



a: Name, Factor, Period, Range
b: EnterN, EnterF, EnterP, EnterR
c: Notify
d: StoreFactor
e: RegisterValue
f: FactorEvidence

Exercise: what else?

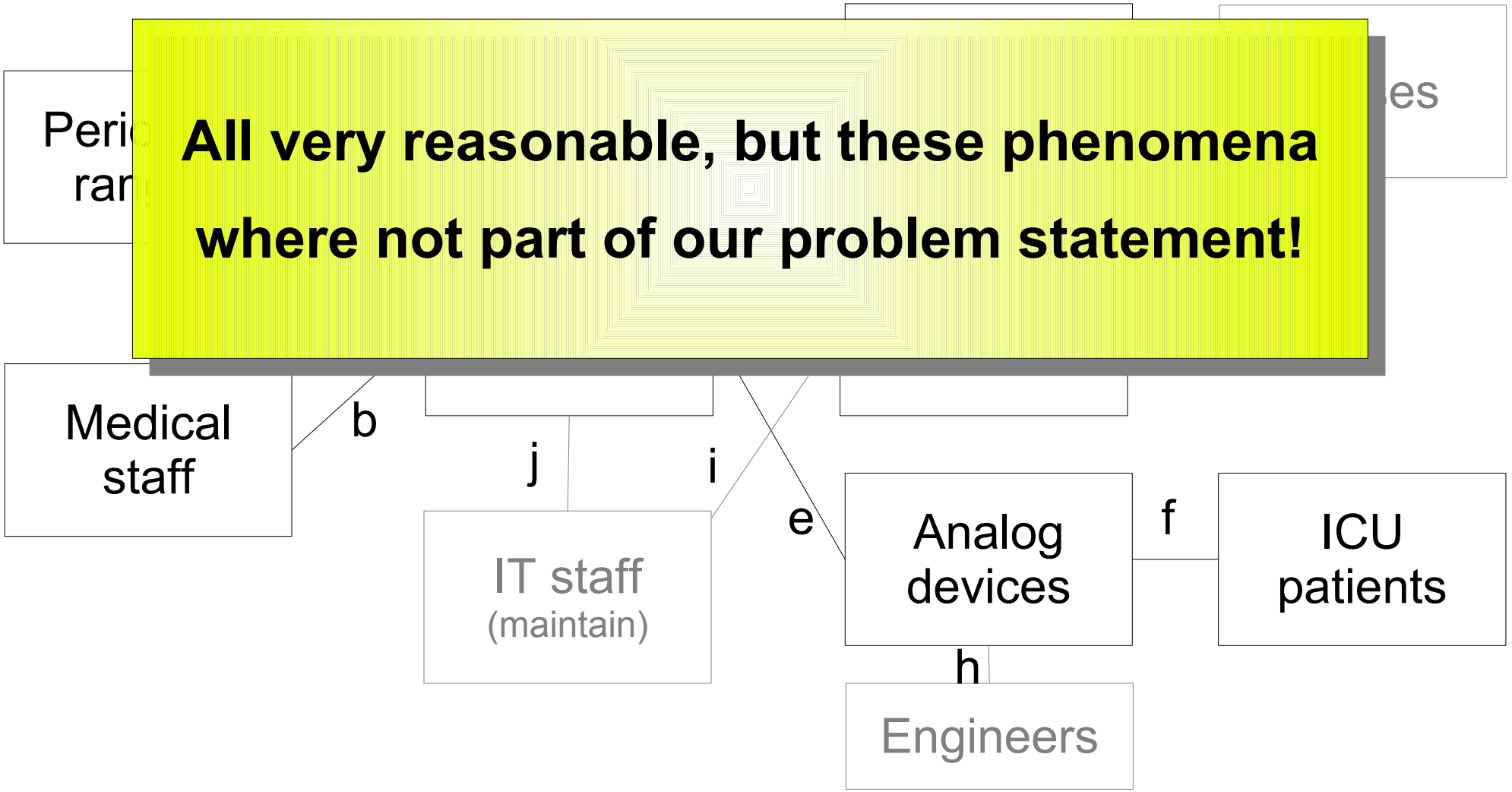
a: Name, Factor, Period, Range
b: EnterN, EnterF, EnterP, EnterR
c: Notify
d: StoreFactor
e: RegisterValue
f: FactorEvidence



Exercise: what else?

a: Name, Factor, Period, Range
b: EnterN, EnterF, EnterP, EnterR
c: Notify
d: StoreFactor
e: RegisterValue
f: FactorEvidence

All very reasonable, but these phenomena where not part of our problem statement!



Exercise: what else?

a: Name, Factor, Period, Range
b: EnterN, EnterF, EnterP, EnterR
c: Notify
d: StoreFactor
e: RegisterValue
f: FactorEvidence

All very reasonable, but these phenomena where not part of our problem statement!

Should they be?

Can we assume them?

Matter for more requirements elicitation and customer interaction

ICU patients