Run-Time Fault Diagnosis for the Grid

October 16th, 2007

Jan Ploski and Wilhelm Hasselbring

OFFIS Institute for Information Technology
TrustSoft Graduate School on Trustworthy Software Systems
Software Engineering Group, University of Oldenburg

GEFÖRDERT VOM

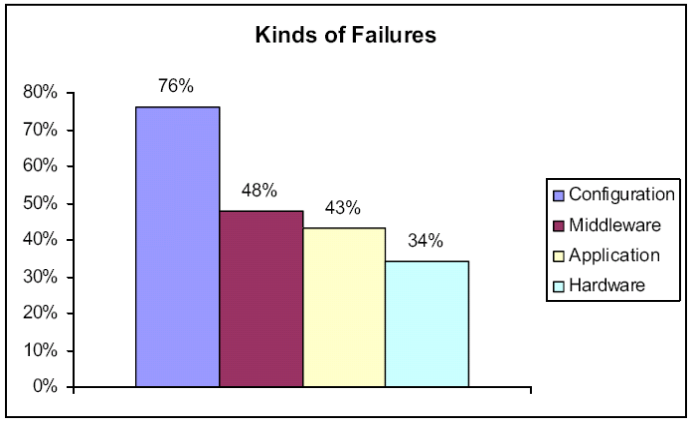
Bundesministerium
für Bildung
und Forschung

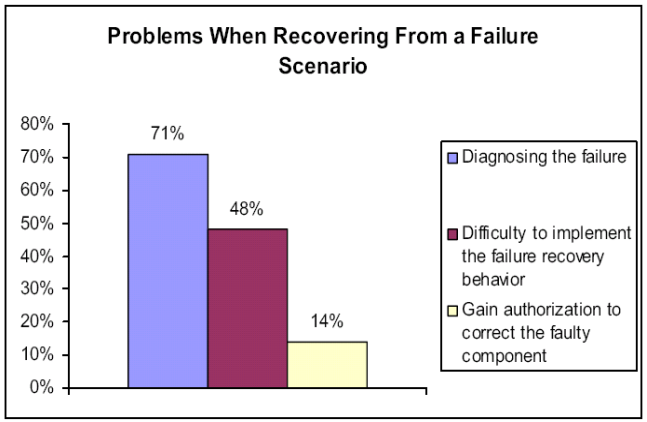
Motivation

A major barrier to widespread acceptance of Grid technology is the difficulty of troubleshooting and debugging Grid applications.
(Tierney, 2004)

In modern services such as e-commerce, telecommuting, VPN, ASP, ***Grid services****, fault localization techniques capable of isolating faults in application and service layers are needed.*
(Steinder, 2004)

Grid components that provide high-level abstractions when working, do expose all gory details when broken.
(Medeiros, 2003)





Source: Grid user survey by Medeiros, 2003

Example of fault diagnosis

- **Faults – original (unpropagated), persistent errors in system state**
- **Repair – removes a fault to prevent failure**

> *The partition with /var/spool/torque on the target execution node was full. Therefore, pbs_mom job output could not be saved. Therefore, no job output could be copied to the submitting node after the job completion. Therefore, no job output was staged out.*
>
> *To fix this problem, make disk space available on the partition containing /var/spool/torque.*

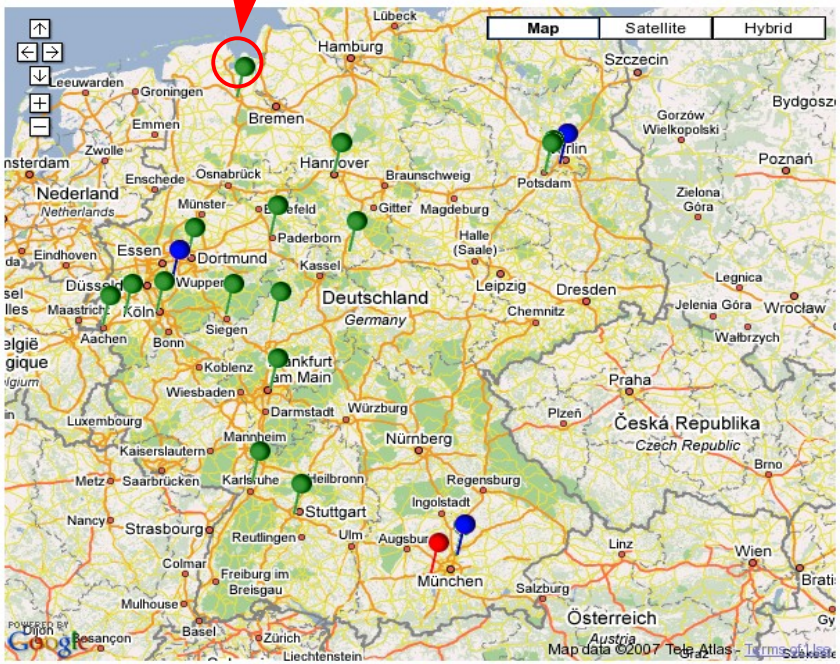
- **Cost: several hours of administrator's and user's time**
- **Troubleshooting tool: strace**
- **Would the diagnosis be faster if it happened again elsewhere?**

Assumptions for our research

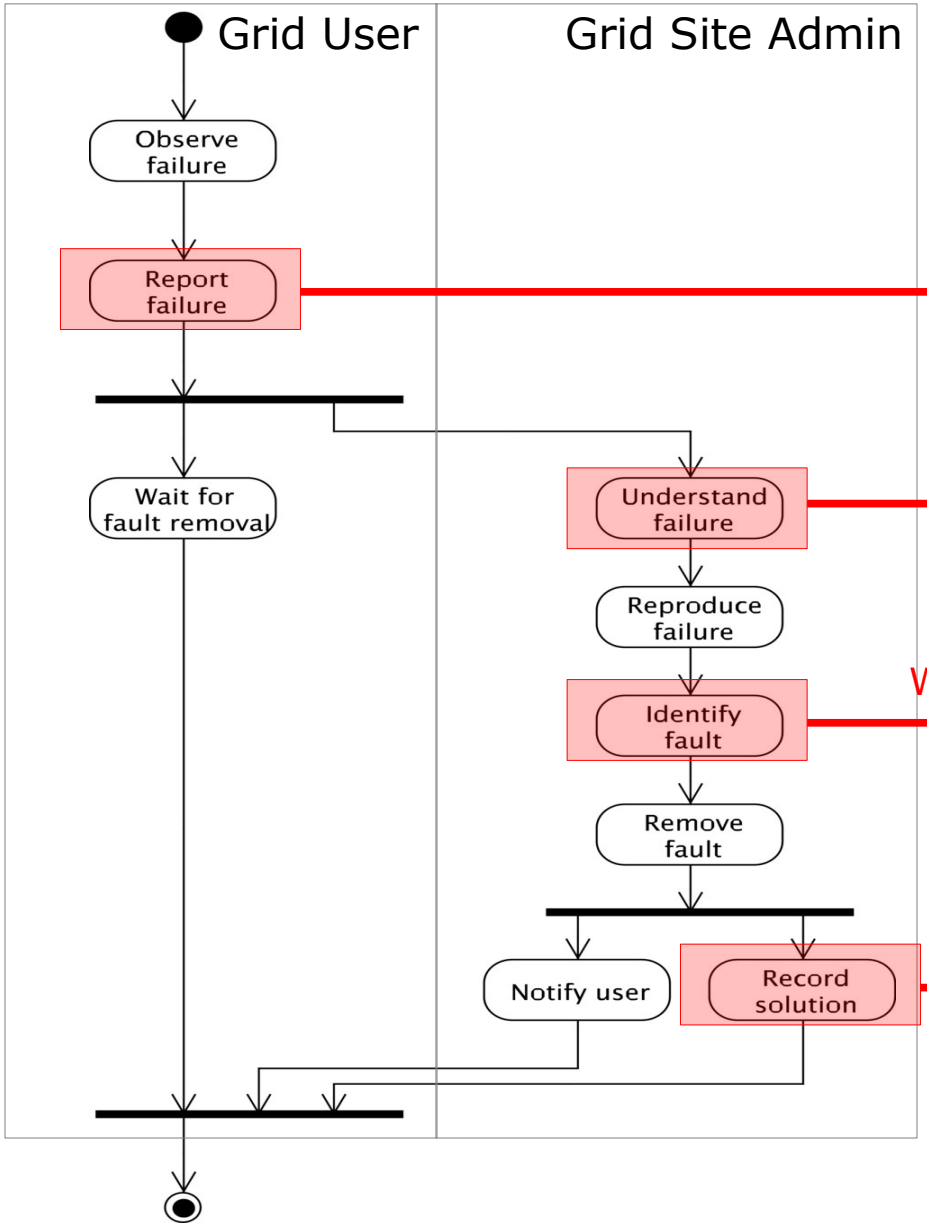
- **Run-time fault diagnosis in Grid environments is difficult/expensive.**
- **The difficulty lies primarily in not knowing what information should be gathered to explain the observed failure.**
- **Faults reoccur across time and across software installations.**
- **The currently used ways of describing faults and failures can be improved to speed up diagnosis.**

Research goals

- **Test our proposed method - event-based run-time fault diagnosis - by implementing it within D-Grid (German Grid).**
- **Test the ability of our models to capture behaviors that have to be reasoned about during diagnosis.**
- **Speed up repeated diagnoses.**



Run-time Fault Diagnosis: State of the art



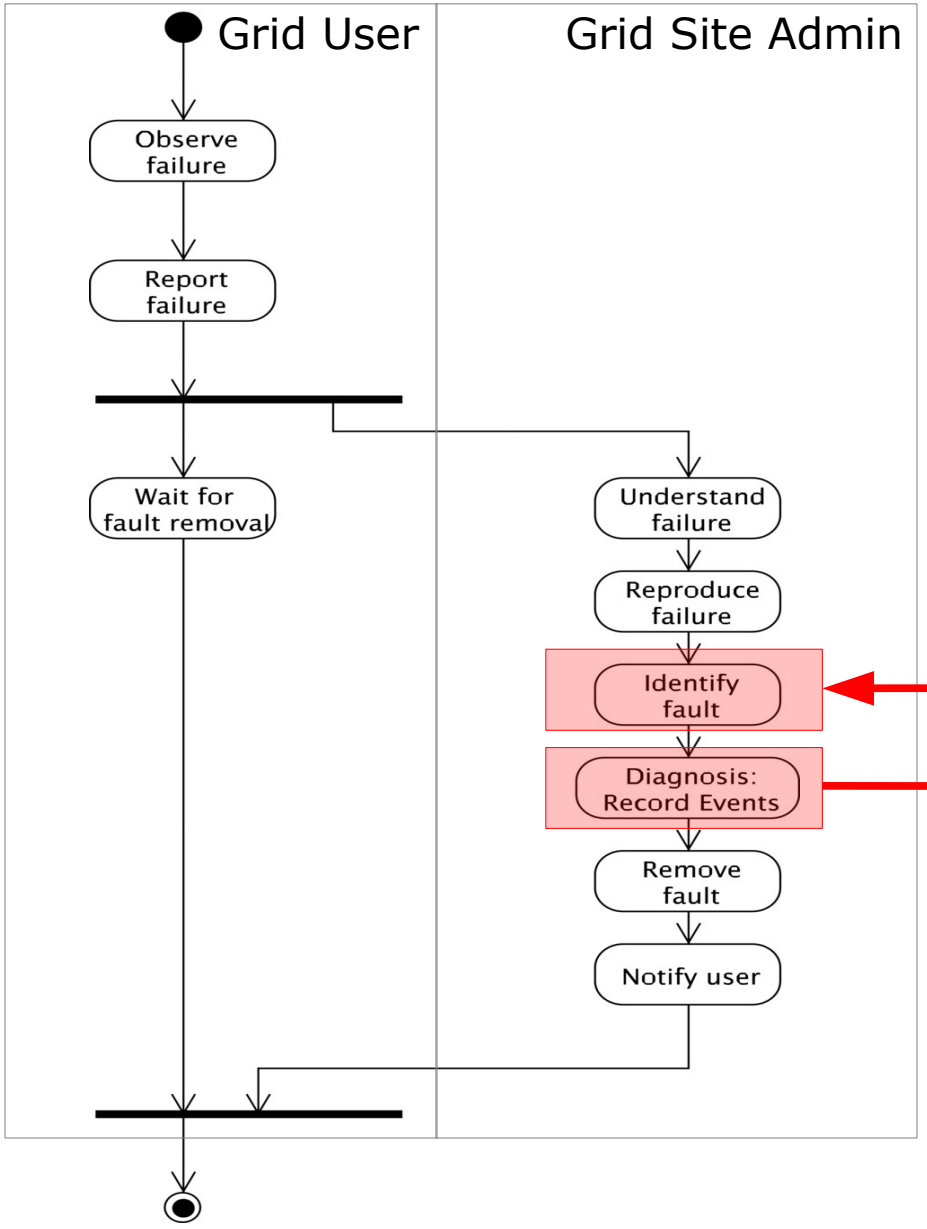
Email or ticket in DGUS

May involve communication with user

With personal experience, Google, prev. tickets

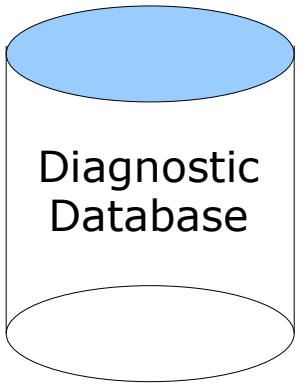
In plain text, optionally

Run-time Fault Diagnosis: Event-Based Approach



Repeated diagnosis: identify a matching case based on the observed events

First-time diagnosis: describe a new *case* in terms of observable *system events*



Shared across Grid sites

Implementing event-based run-time fault diagnosis

WP1
Observability

WP2
Representation

WP3
Automation

Implementing event-based run-time fault diagnosis

WP1 Observability

WP2 Representation

WP3 Automation

- **Define basic terminology**
  - Fault, failure, repair
  - Run, event, event sequence, trace
  - Diagnosis, diagnostic case
  - ...
- **Select event types relevant for diagnosis**
- **Improve component observability through instrumentation (technology-specific)**
  - No observable events = no diagnosis
  - Major technical and semantic challenges due to heterogeneity!

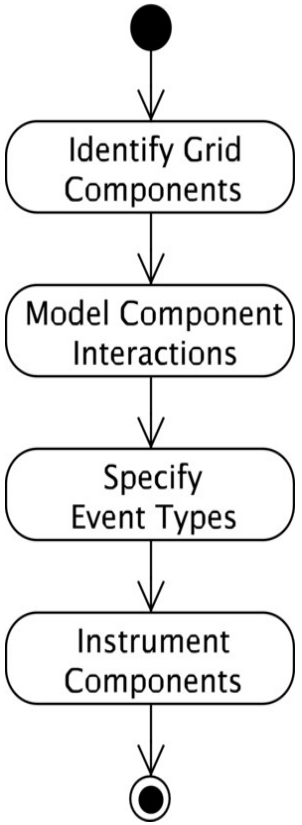
Implementing event-based run-time fault diagnosis

WP1 Observability

WP2 Representation

WP3 Automation

- **Input**
  - Narrative descriptions of actual diagnostic cases
  - Middleware documentation
  - Experiments
  - Exception handling theory
- **Results**
  - Specified observable event types
  - Instrumented Grid middleware components – in WISENT cluster
  - Uniform format of event logging



Implementing event-based run-time fault diagnosis

WP1
Observability

WP2
Representation

WP3
Automation

- **Define a language for representing diagnostic cases**
- **A diagnostic case describes a sequence of observable events, postulating their occurrence or non-occurrence in some order**
- **Specify how diagnostic cases are stored and retrieved from the diagnostic DB**

Implementing event-based run-time fault diagnosis

WP1
Observability

WP2
Representation

WP3
Automation

- **Input**
  - Sample event traces captured by instrumentation developed in WP1
- **Results**
  - Syntax and semantics of a language for describing diagnostic cases
  - Implementation of the diagnostic DB

Implementing event-based run-time fault diagnosis

WP1
Observability

WP2
Representation

WP3
Automation

- **Develop a middleware component which**
  - selectively enables instrumentation
  - compares actual system behavior to the contents of the diagnostic DB
  - suggests repair actions on match
- **Develop a feasible user interface for administrators performing "repeated diagnoses"**
- **Enable collection of frequency data about actual fault occurrences**

Summary: expected contributions

WP1
Observability

WP2
Representation

WP3
Automation

- **A method for reducing the impact of faults in a Grid by improving run-time fault diagnosis**
- **Recommendations for design and implementation practices that support run-time fault diagnosis**
- **Evaluation of the ability to perform automated diagnosis of known faults**

WP1 Observability

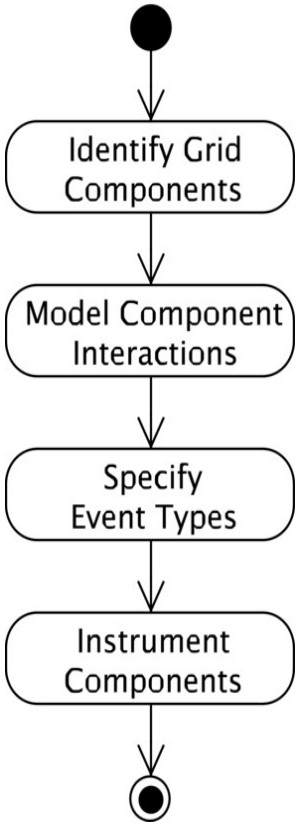
Source: http://www.midkentwater.co.uk/household/problem/fixing%20a%20leak.htm

The ability of a diagnoser to detect the occurrence of events that indicate the location of an error.

WP1 Observability – Modeling & Instrumentation

- **Identify the software components of a Grid site**
  - Focus on events reported by these components
  - Component instances are sometimes created and destroyed dynamically
  - Real diagnostic cases, documentation, log files guide component identification

- **Instrument components to improve observability**
  - Log scraping provides one source of recorded events
  - Tracing selected system/library calls (e.g. file and network I/O) is another source
  - Event traces must have a uniform format for evaluation
  - Event types must include context information to support cross-referencing (e.g. process, job, WS resource ids)
  - The instrumentation should not require modifications at source code level.



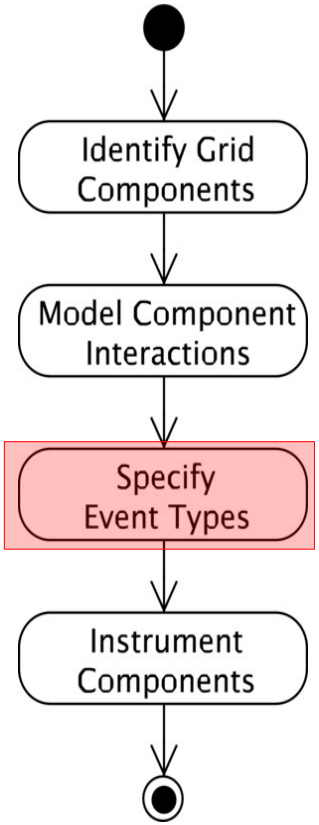
WP1 Observability – Exceptional Events

- **Errors are incorrect parts of system state.**
- **Events are state transitions**
  - Some emit messages thanks to instrumentation
- **Which events most likely indicate an error?**

Exceptions that occur early in the observed run.

Exceptions (return codes, …) are a built-in classification mechanism provided by programming languages and/or execution environments.

- **However, exceptions do not always indicate errors**
  - e.g. handled exceptions, misuse
- **Need to combine knowledge about their occurrence with contextual information – other events**



First-time diagnosis – performed by an expert

- **Generate plausible error hypotheses**
  - compare the non-working configuration to a working one (if available)
    (cf. Delta Debugging, Zeller 2005)
  - recall prior solved cases with "similar" symptoms
- **Test the error hypotheses**
  - assume their truth
  - infer necessary consequences
  - check whether consequences actually occur
- **Exonerate components**
  - compare actual vs. expected outputs (if known)
  - analyze (im)possible data and control flows

First-time diagnosis, cont.

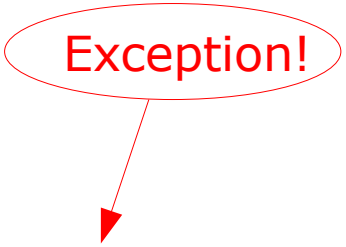
- **Result of a successful first-time diagnosis: a natural language explanation of the error-failure link, along with the required repair action:**

  The partition with /var/spool/torque on the target node was full. Therefore, the job output could not be saved. Therefore, no job output could be copied to the submitting node after the job completion.

  To fix this problem, make disk space available on the partition containing /var/spool/torque.

- **Such a description is then (manually) translated into a statement about a sequence of observable events in the system.**

First-time diagnosis, cont.



1. PBS job submitted to node <n>
2. syscall write in process pbs_mom on node <n> returns ENOSPC
3. job terminates
4. job output file does not exist or has old timestamp

- **The formal "event language" for describing diagnostic cases depends on the prior instrumentation of software components:**
  - only the actually observed events can be referenced
  - abstraction from event instances to event types is a must (e.g. syscall write in pid 1234 -> syscall write in pbs_mom process)
- **The characteristic event sequence is saved in a central diagnostic database together with recommendations of repair actions (in natural language).**
- **A repeated diagnosis is performed to test recall capability.**

Impact of heterogeneity

- **Grids are shockingly heterogenous.**
- **The only common denominator seems to be Linux.**
  - We don't even agree on a distribution →kernel version
- **Apart from that, everything goes**
  - Compiled C(++, Fortran) code, statically or dynamically linked
  - Java
  - Shell scripts
  - Scripting languages: Python, Perl, Ruby, <*insert favorite*>
  - Specialized language interpreters
  - Commercial software without source code

- **Unfortunately, the concept of (high-level) exceptions varies across all these run-time environments.**
- **An external instrumenter must thus...**
  - Use different instrumentation techniques
  - Prefer "easy" instrumentation →low-level events, interfaces

Low-level instrumentation techniques for Linux

- **Library call interception with LD_PRELOAD**
  - Only works for dynamically linked code
- **Static ELF instrumentation using the ERESI framework**
  - Not supported on AMD64 architecture
  - Also flaky on IA32
- **kprobes/Systemtap**
  - Can only instrument kernel-space code
  - (Today) Difficult to install (e.g. kernel upgrades, special debuginfo package), poorly documented
  - (Today) Alpha-quality - may cause system crashes (+ other security issues!)
  - (Today) Not compatible with Xen
- **Strace, ltrace, gdb and other *ptrace*-based tools**
  - Work as advertised
  - Inefficient by design (context switching)
- **uprobes, utrace-based tools**
  - Can instrument user-space code
  - Alpha-quality, undocumented, "not yet" (x86_64 in progress)

Cross-referencing, event correlation

- **Apart from technical difficulties, low-level instrumentation techniques pose conceptual challenges:**
- **Matching high-level events with low-level events is difficult**
- **Example:**
  - Condor job id →Process id on submission machine →Globus job id →RFT transfer id →Process id #1 →Process id #2 →. . . →Library call →System call
  - How can we trace back a given system call to the initial Condor job id?
  - Yet this sort of mapping is required to avoid collecting and/or examining huge amounts of irrelevant data
- **Hope for time-based event correlation, automated filtering**

WP2 Representation

- **Given an event trace from WP1, how do we select the error-induced events?**
- **Need a language for expressing allowable event orderings, occurrence, non-occurrence, event correlation, generalization**
- **Should it support less-than-definite statements?**
  →probabilistic modeling
- **Suggestions?**

WP3 Automation

- **Allow the user to specify the begin and end of a "run" during which events should be observed and a failure occurs.**
- **Let the diagnostic tool select which events to observe based on the database of known diagnostic cases.**
- **Let the diagnostic tool match against the "most likely" diagnostic case and suggest repair actions.**

Supplementary slides begin here

Faults and causality

- **A fault is the "adjudged or hypothesized cause of an error"** (Avizienis 2004)
- **But what is a *cause*, anyway?**
- **In our approach, the fault is "the part of system state altered by repair", that is, the actually removed error.**
- **For this reason, we avoid the term *fault* altogether.**
- **Subjective and confusing?**

Causality vs. probability (Pearl 2001)

- **Probability...**
  - What is the (likely) system state...
    - given that we know this-and-that?
  - Reasoning about the current state ("static view")
- **Causality...**
  - What will (likely) happen...
    - given that we know this-and-that
    - if we change the system so-and-so?
  - Reasoning about the effects of interventions ("dynamic view")

Causality - definition

- **Causality is…**
  - Intuitively understood and invoked by everyone
  - Not easy to define formally (in general)

- **Example:**
  - Let *A*, *B* be propositions concerning the occurrence of two subsequent system states
  - Let *do(A)* be the proposition: "The system state will be manipulated to make A true."
  - Define causality as follows:

    $P(B|do(A)) > P(B)$ learning that *A* will be manipulated to become true makes us predict that *B* is more likely to occur

- **Reasonable?**

Causality - the preemption paradox

- **A – the security patch is installed**
- **B – a break-in will occur during the next month**

- **We predict… *P(B|do(A)) < P(B)***
- **Later we find out that**
  - the patch actually created a new security hole
  - hackers successfully exploited that hole to break in

- **Q: Did the installation of the security patch cause the security breach?**
- **Yes, according to common sense. No, according to our definition.**
- **But! Our information has changed (*C* – patch was bad), so we have**

P(B|do(A)) < P(B)
P(B|do(A)C) > P(BC) the "had we known" loophole rescues the definition

Causality - conclusion

- **Causality, like probability, *is* subjective...**
- **...in the sense that it is always conditioned on some information.**

- **So what?**
  - not a great concern for "practical" cases
  - consider pitfalls on a case-by-case basis
  - be explicit about prior information
    a "best practice" (not widely followed)

Probability theory – a gun without ammo?

- **Probability theory defines the normative formal rules of inductive reasoning.**
- **Given information in form of probability distributions for some logically related variables, it allows deriving some interesting statements concerning values of other variables (which might be hidden, but verifiable through experiments).**

- **However, probability theory does not tell us**
  - Which variables to include in the model
  - How to convert informal knowledge into probability distributions

- **The former task is accomplished by "informed guesses"**
- **The latter is objectively possible only for**
  - Information in form of frequency distributions
  - Testable constraints on observable data (via MAXENT method)

Subjective probabilities

- **Probability distributions may also be used to represent opinions ("expert knowledge").**
- **Here, we assume that any probability distribution is allowable in a model, as long as no inconsistencies arise ("expert does not disagree with himself").**
- **Avoiding inconsistency is easy (guaranteed within a Bayesian Network model).**
- **Obviously, this approach has some problems**
  - Model variables must be selected somehow (as usual).
  - A model based on opinions rather than on verifiable experimental data is only as good as these opinions. (Could be consistent, but wrong.)
  - Even though opinions are supposedly based on data, it is impossible to test their validity directly.
  - Testing that the original opinions were correctly incorporated into a probabilistic model is difficult.

Subjective probabilities, cont.

- **Scientifically unwarranted opinions do matter in the "real world".**
- **Many domains, including engineering, favor human expertise over scientific data because of easier accessibility.**
- **The expertise, to be called such, should rest upon data.**
- **But in many cases the data assimilation process is impossible to formalize.**