A Taxonomy for Classifying Runtime Verification Tools

Ylies Falcone[1] Srdan Krstic[2]
Giles Reger[3] Dmitriy Traytel[2]

[1]Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, 38000 Grenoble, France

[2]Institute of Information Security, ETH Zürich, Switzerland

[3]University of Manchester, Manchester, UK

November 13, 2018

Authors

Introduction

Highlights

- Taxonomy of RV techniques with (around) 20 features
- Categorisation of 20 tools (with 30 more identified)

This Talk

- Introduce context and approach taken
- Use two tools to take a detailed look at the Taxonomy
- Overview the full classification
- Discuss some observations and future work

Acknowledgements

- Martin Leucker for early discussions on taxonomy & representation.
- Working Groups 1 and 2 of COST Action ARVI IC1402
- Participants of Dagstuhl seminar 17462.

Motivation

We have lots of tools (and growing)

What are the key concepts behind them, how do they compare?

Interesting from a practical perspective

- What tool should I use?
- What tools should I compare to?
- How should a competition/challenge be organised?

Interesting from a theoretical perspective

- What is the space of exploration?

As a community we've talked about this a lot but we've been held back by wanting to get it perfect. Our solution is not perfect but a good starting point.

Previous Ideas on Classification

The Standard Online/Offline, Inline/Outline

- Online - whilst system running
- Offline - after system ran
- Inline - embedded in system (somehow)
- Outline - separate from system

Other Terms Used

- Synchronous vs Asynchronous
- Event-triggered vs Time-triggered

All captured by our taxonomy.

Previous Ideas on Classification

1st RV Competition in 2014

Table 5 Summary of features of the tools

Participating tool	User-enabled	Built-in	Propositional events	Parametric events	Automata-based	Logic-based	Regular Expressions-based	Logical-time	Real-time	Own instrumentation	Relies on AspectJ	Relies on another technique	C programs	Java programs	Traces	Time triggered	Event triggered
	Input requirement specification									Instrumentation			Monitored systems			Monitoring mode	
RiTHM	✓		✓	✓		✓		✓		✓		✓	✓		✓	✓	✓
E- ACSL	✓	✓	✓	✓		✓				✓			✓				✓
RTC		✓								✓			✓				✓
Larva	✓		✓	✓							✓			✓		✓	✓
jUnit^{RV}	✓		✓	✓	✓				✓	✓		✓		✓			✓
JavaMop	✓		✓	✓	✓	✓	✓	✓			✓			✓			✓
MonPoly	✓		✓	✓		✓		✓	✓						✓		✓
STePr	✓		✓	✓											✓		✓
MarQ	✓		✓	✓	✓			✓			✓			✓	✓		✓

Previous Ideas on Classification

ParTrap paper presented yesterday

Table 1. Comparison of ParTraP with several temporal specification languages

Language	Para-metric	Comp. values	Quan-tifica-tion	Ref. past data	Wall-clock time	Style
Dwyer's patterns [13], Propel [22], LTL_f [6], CFLTL [20]	✗	n/a	n/a	n/a	✗	decl.
RSL [21], Salt [7], $TLTL_f$ [6]	✗	n/a	n/a	n/a	✓	decl.
Eagle [2]	✓	✗	global	✗	✓	decl.
Stolz's Param. Prop. [23]	✓	✗	local	✗	✗	decl.
FO-LTL^+ [14]	✓	✓	local	✗	✓	decl.
MFOTL/MONPOLY [5,11]	✓	✗	global	✓	✓	decl.
JavaMOP [17]	✓	✗	global	✓	✗	mixed
QEA/MarQ [1,19], Mufin [12]	✓	✗	global	✓	✗	oper.
Ruler [4], Logfire [15]	✓	✗	n/a	✓	✗	oper.
LogScope [3]	✓	✓	global	✗	✗	mixed
ParTraP	✓	✓	local	✓	✓	decl.

Previous Ideas on Classification

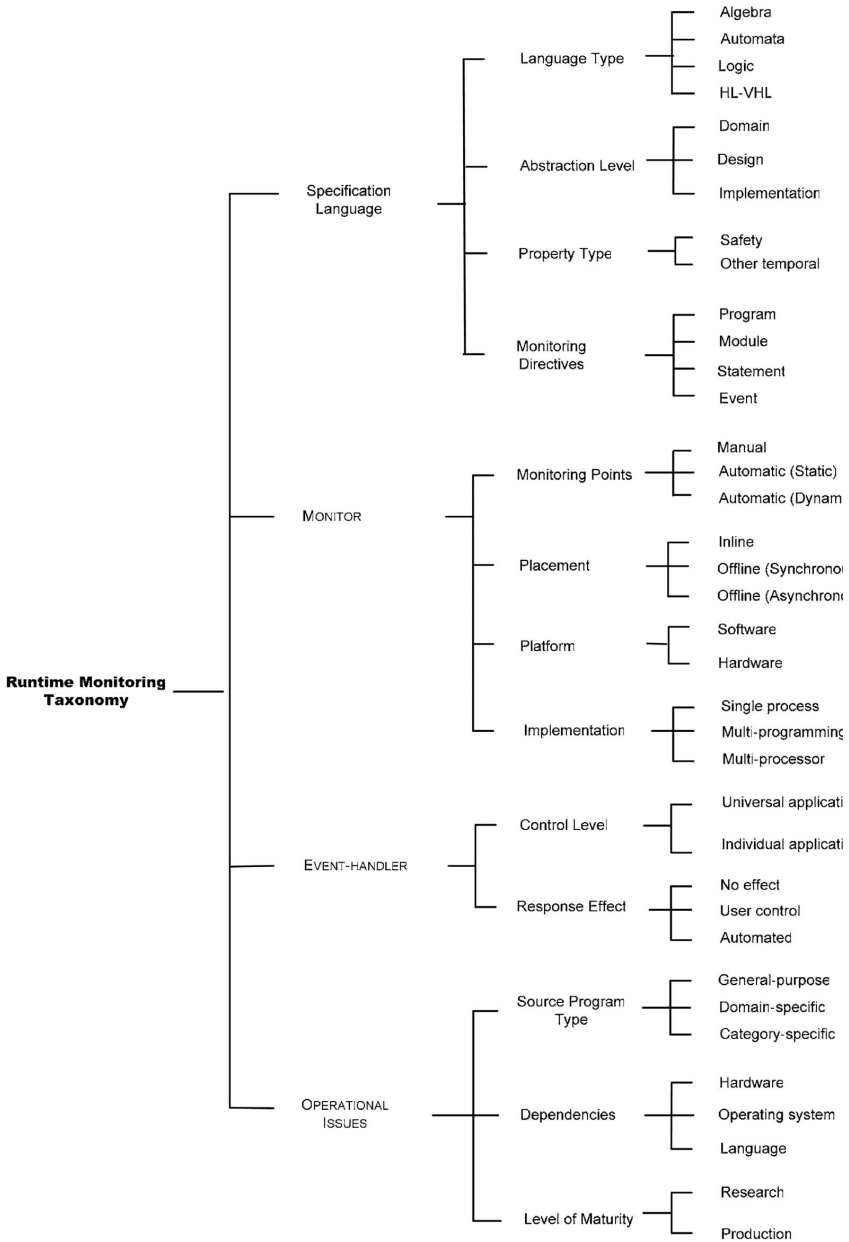
Taxonomy of Delgado et al. (2004)

Four top-level concepts:

1. Specification - focussing on language, class (e.g. safety) and abstraction
2. Monitor - instrumentation and placement
3. Event-Handler - kinds of verdicts/results
4. Operational Issues - maturity of tool, target language, dependencies

Taxonomy was very operational and did not go into much detail on specification language and notions of placement were high-level.

Previous Ideas on Classification



Previous Ideas on Classification

Tool	Language Type				Abs. Level			Property Type		Monitoring Directives			
	Algebra	Automata	Logic	HL/VHL	Domain	Design	Impl.	Safety	Other Temporal	Program	Module	Statement	Event
ALAMO				×			×	×				×	×
ANNALYZER				×		×	×	×		×	×	×	
ANNA CCS				×		×	×	×		×	×	×	
APP				×		×	×	×			×	×	
BEE++				×			×	×					×
DB ROVER			×		×	×	×	×	×	×	×	×	×
DYNAMICS			×				×	×					×
FALCON				×	×		×	×				×	
JASS	×			×	×	×	×	×	×	×	×	×	×
JPAX	×		×				×	×	×			×	×
JRTM			×		×		×		×				×
MAC			×	×	×		×	×	×		×	×	×
MOP	×	×	×	×	×	×	×	×	×	×	×	×	×
NON-INTER				×			×	×	×		×	×	
PMMS			×			×	×	×		×		×	×
RAC			×	×		×	×	×			×	×	
REQMON			×		×		×	×	×	×	×		
SENTRY			×	×			×	×	×	×		×	
T ROVER			×		×	×	×	×	×	×	×	×	×

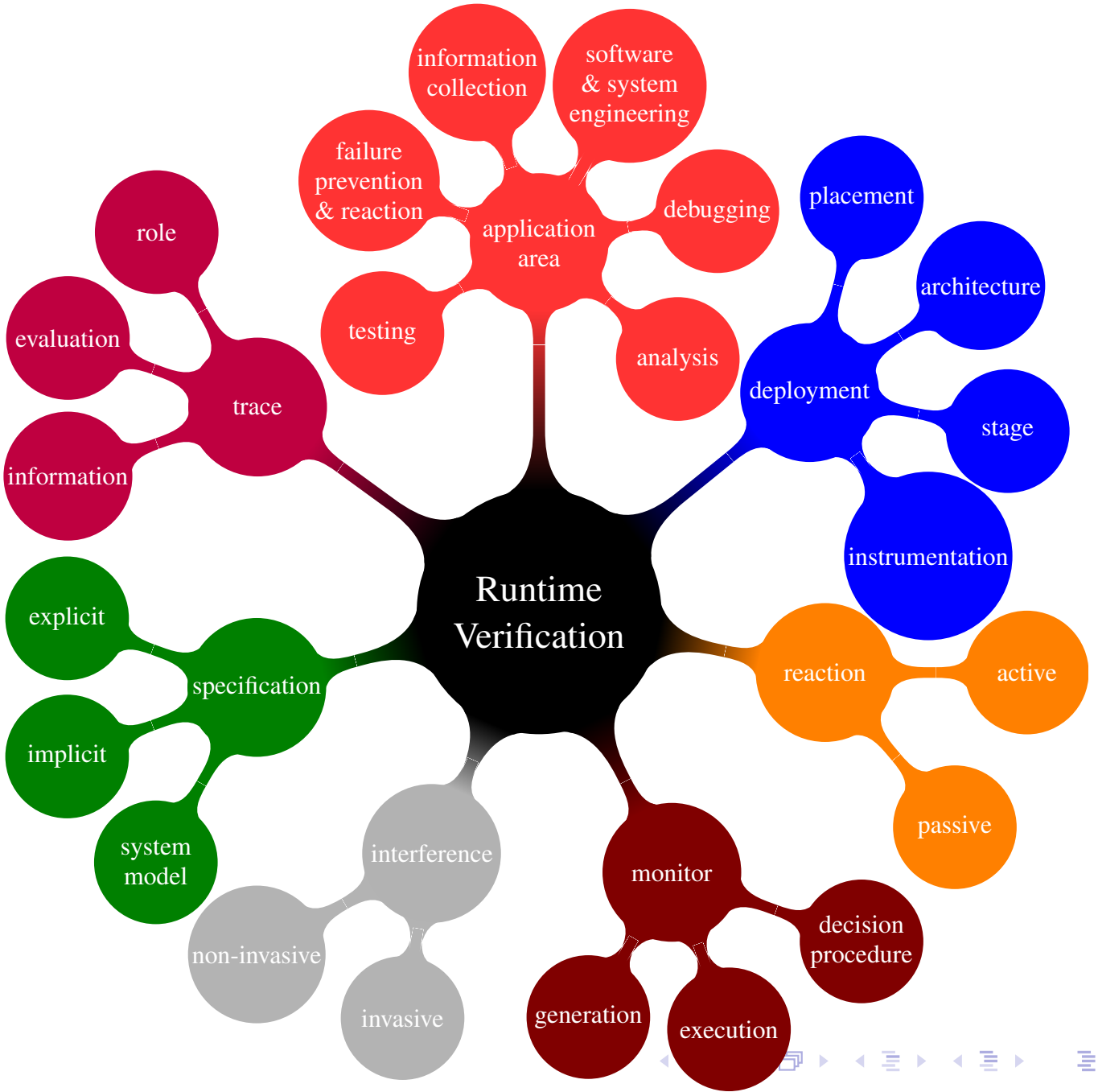
Previous Ideas on Classification

Teaching Runtime Verification by Martin Leucker (2015)

Methodology (counter-example guided refinement)

0. Starting point from discussions/brain-storm
1. Attempt to classify tools
2. If issues with taxonomy GOTO 1
3. Attempt to concisely present and explain taxonomy
4. If issues with taxonomy GOTO 1

Our Taxonomy: The Big Picture



Two Tools

To help us explore the taxonomy we'll look at how two tools are classified with respect to it.

MarQ [Reger et al]

Monitoring at runtime with QEA, based on parametric-trace slicing. The tool is implemented in Java and can read in trace files or being called at runtime, usually but not necessarily, by AspectJ.

Monpoly [Basin et al]

Monitors metric first-order temporal logic (MFOTL) and introduced in the context of checking compliance of log files/event-streams. Now being used in the context of Big Data monitoring.

Specification

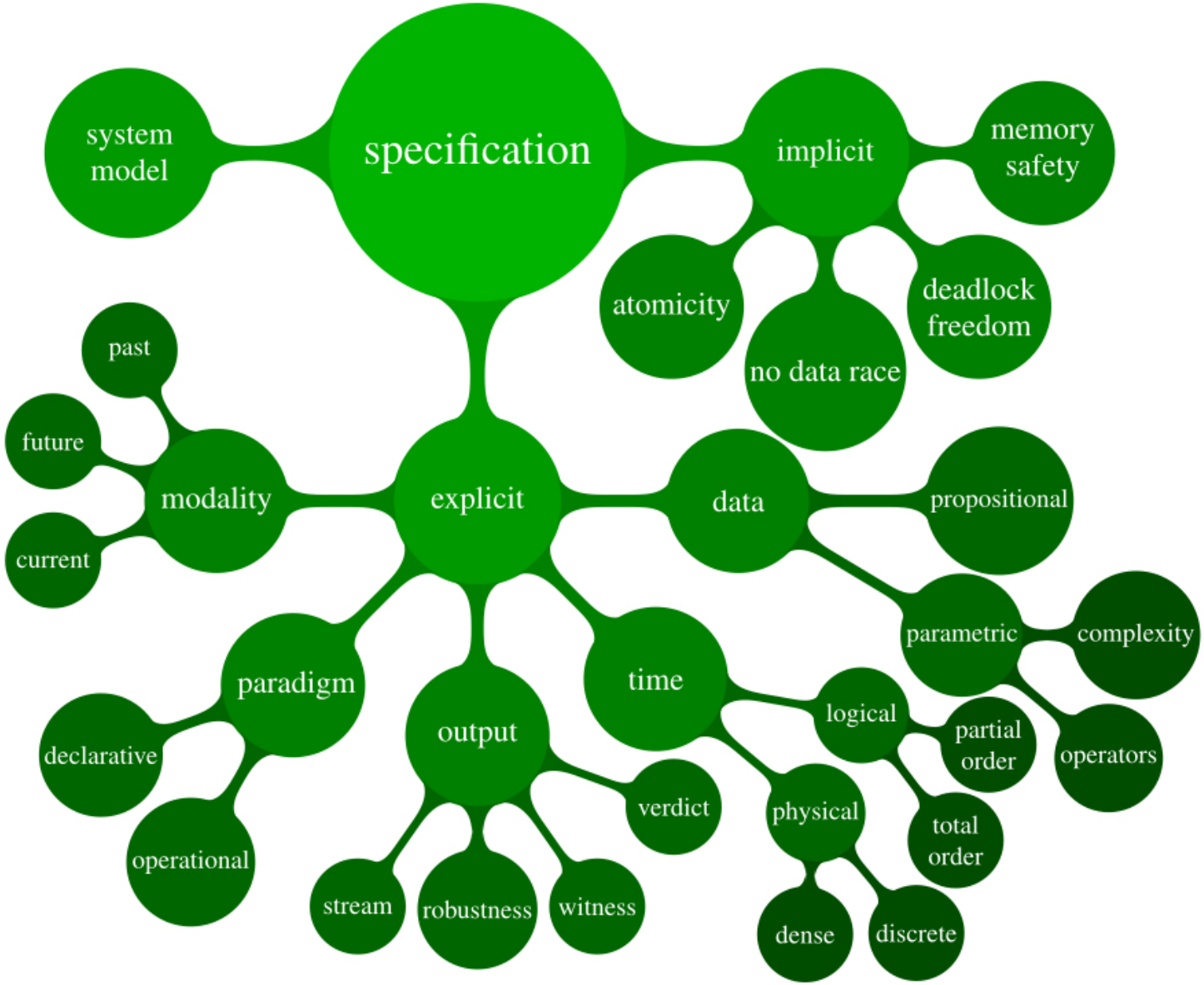
A specification indicates the intended system behavior (property), that is what one wants to check on the system behavior.

A specification exists within the context of a general *system model* i.e., the abstraction of the system being specified.

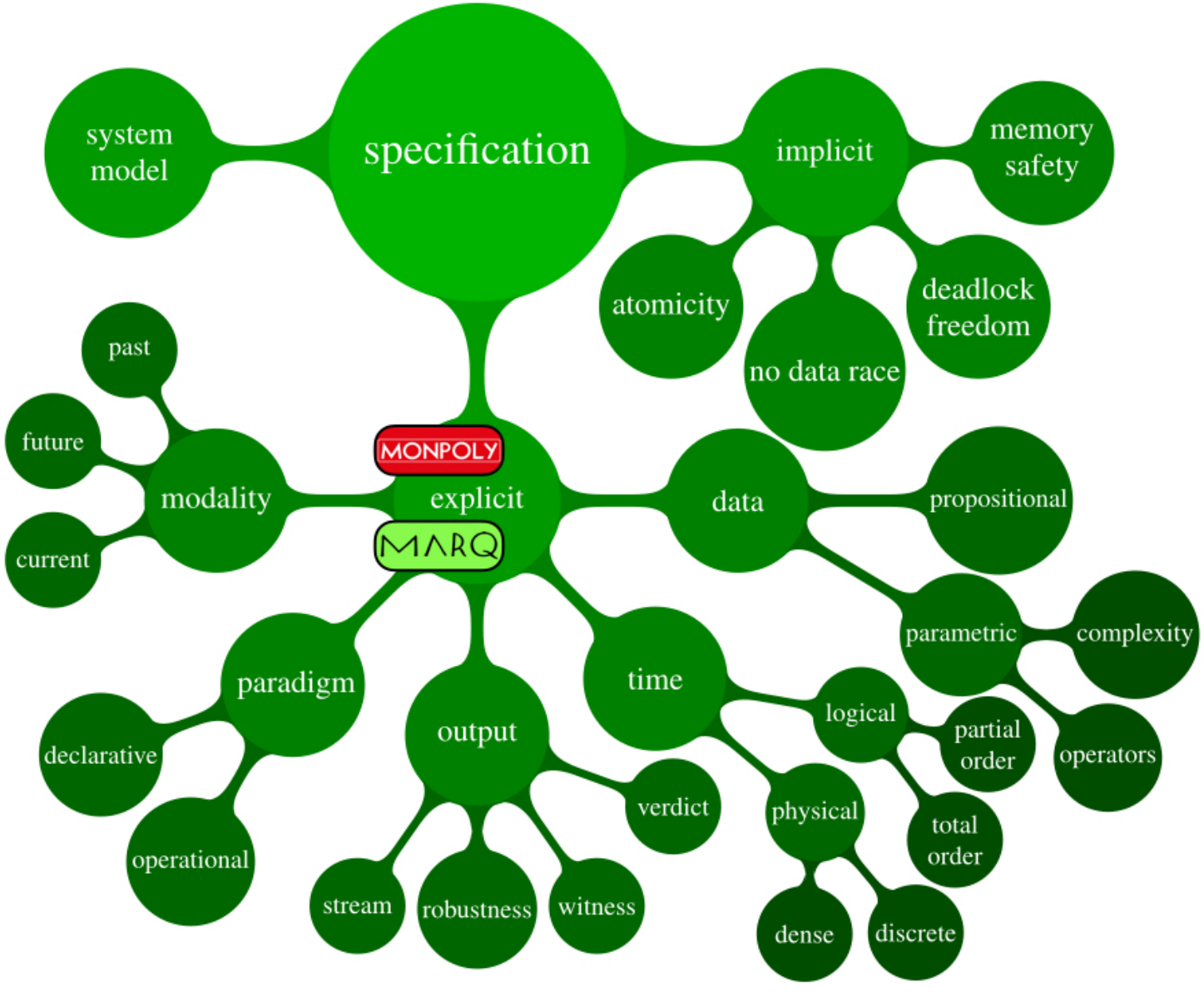
A specification can be implicit or explicit

For explicit specifications we care about properties of the specification language, including the structure of models (traces) in that language

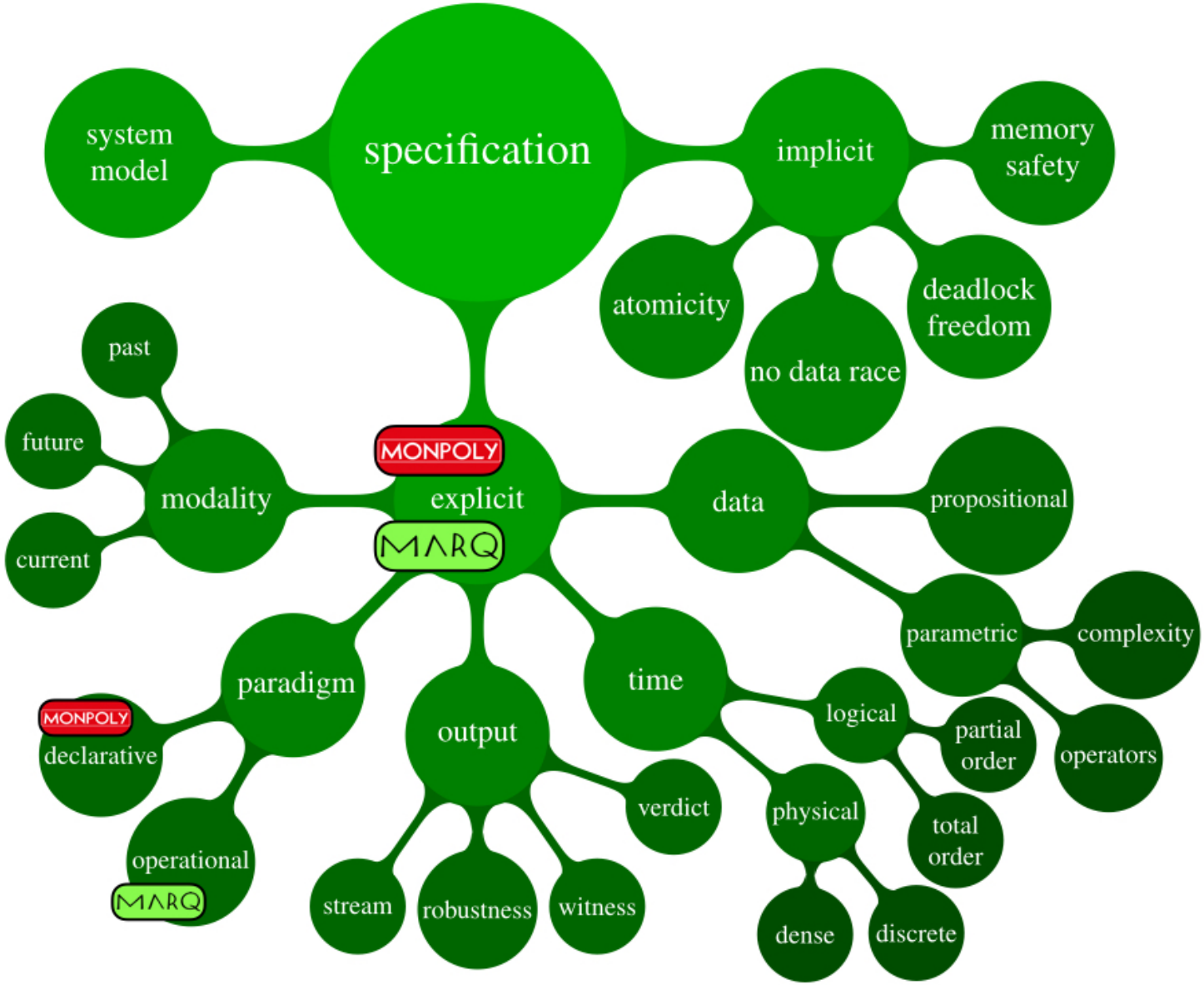
Specification



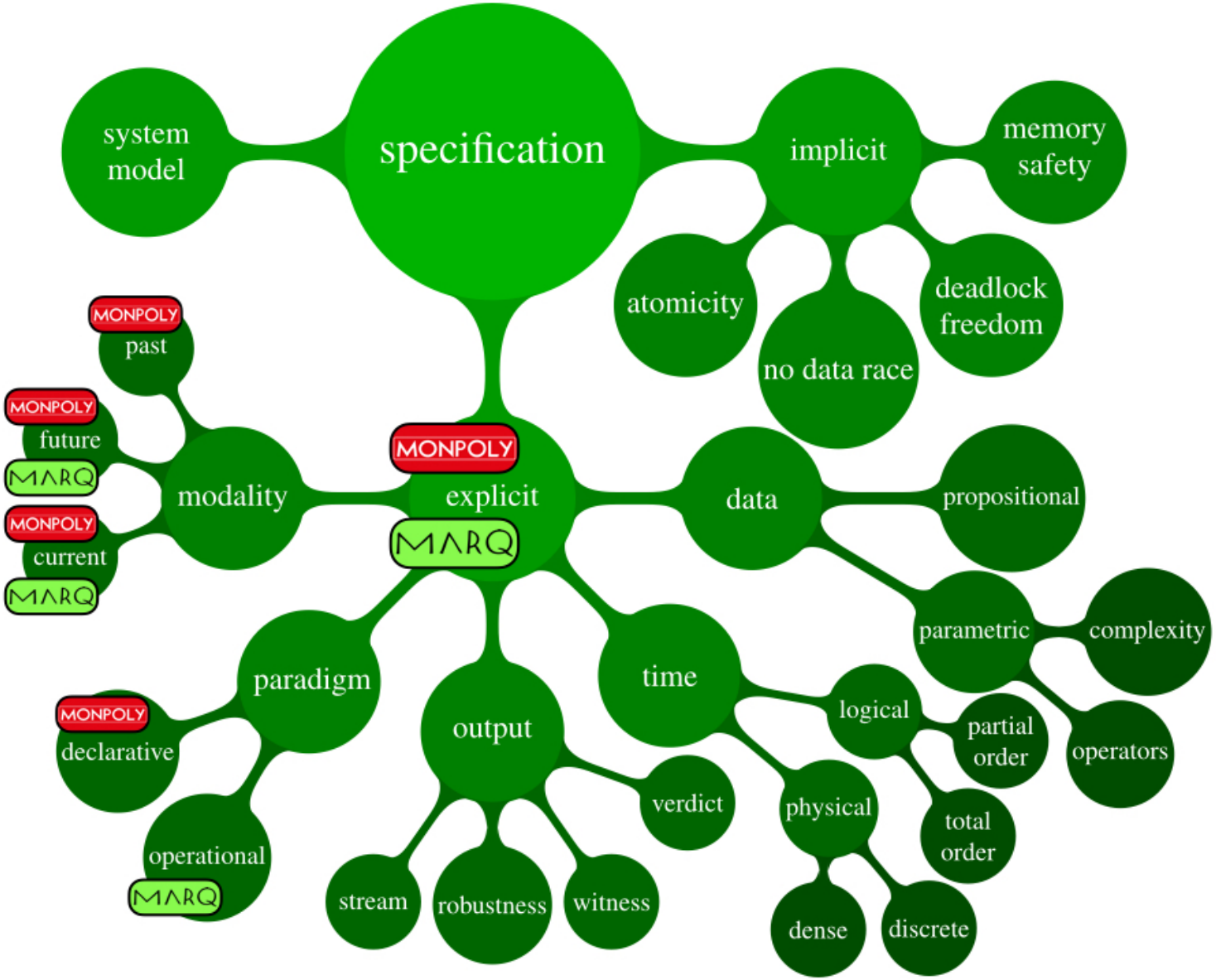
Specification



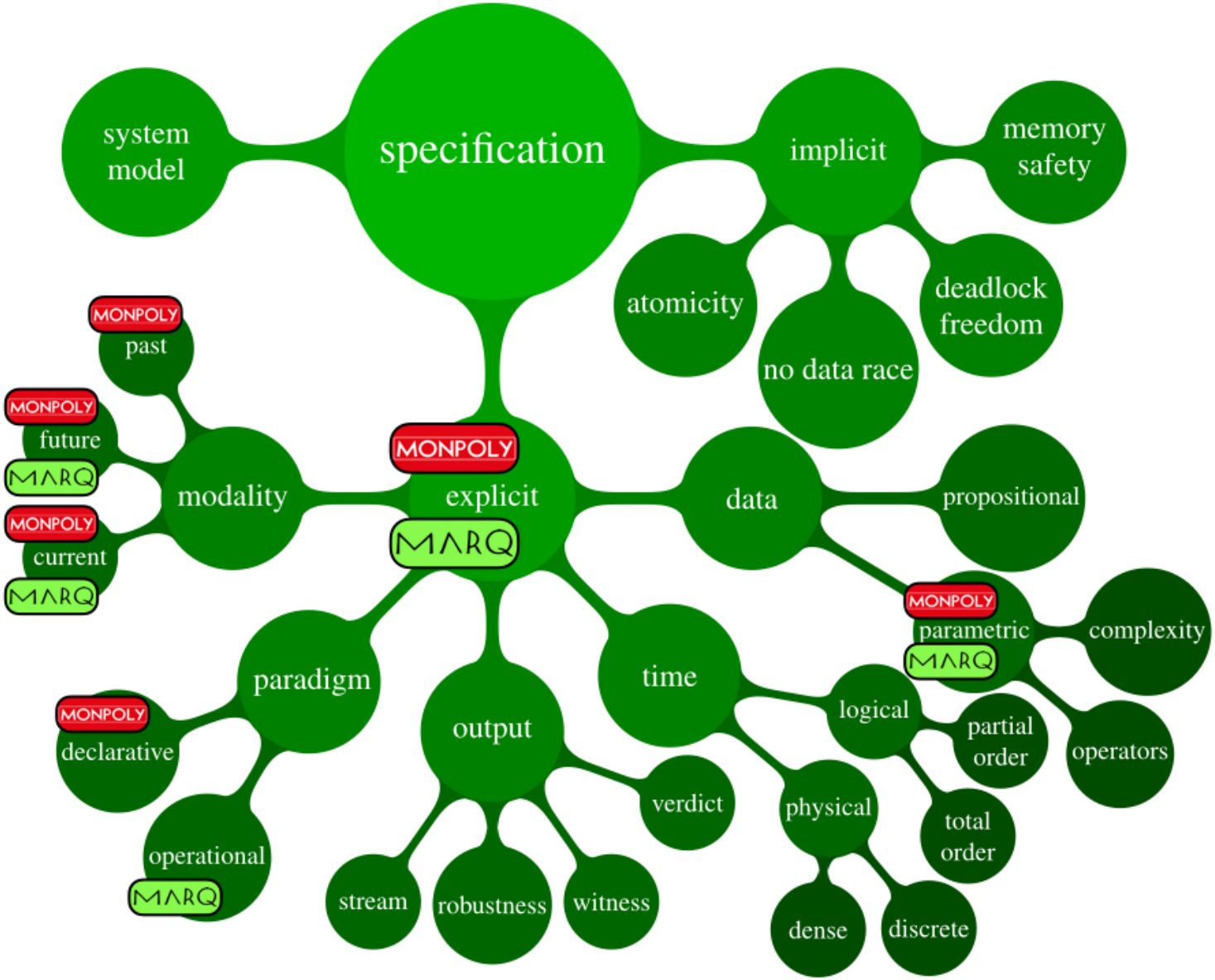
Specification



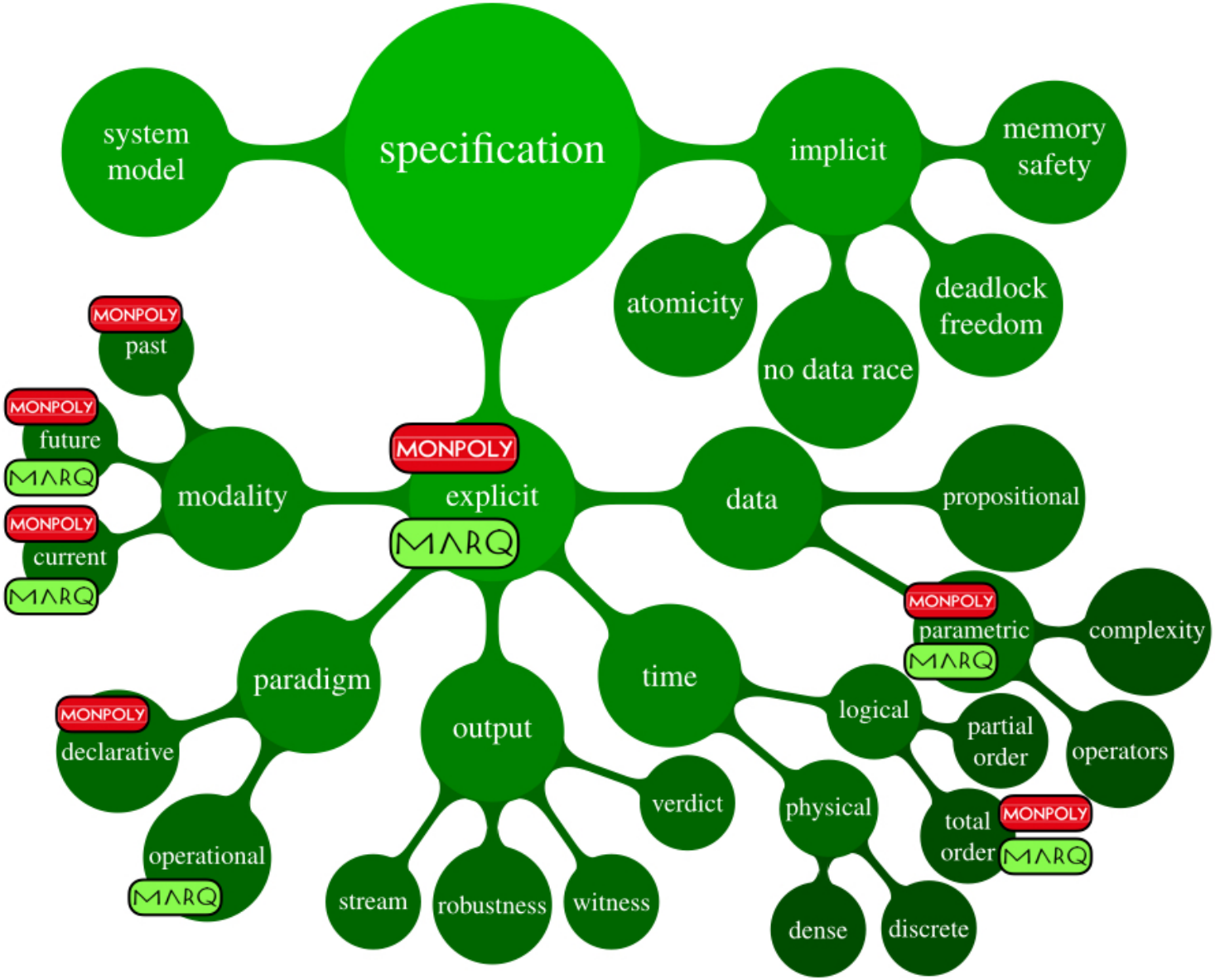
Specification



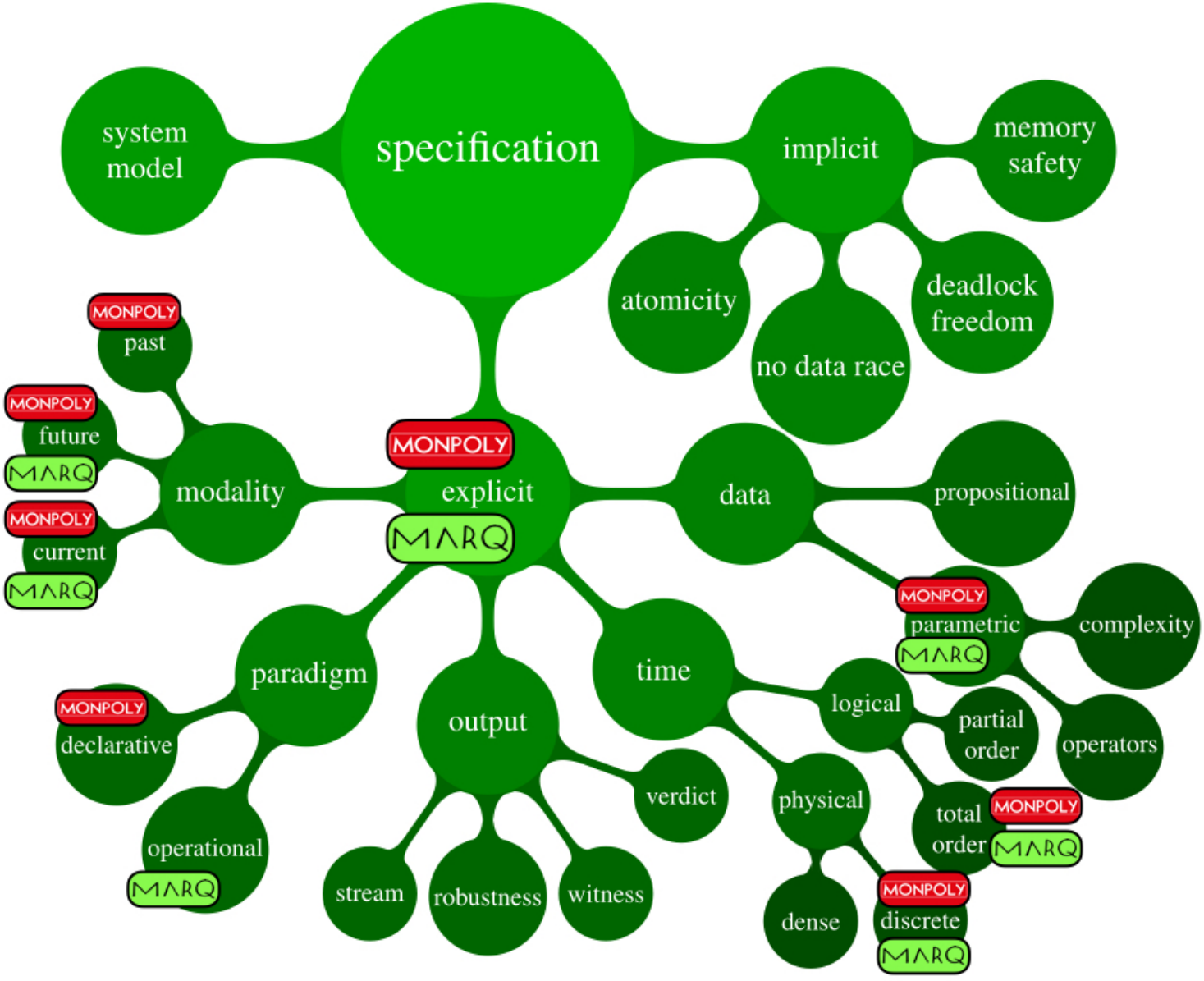
Specification



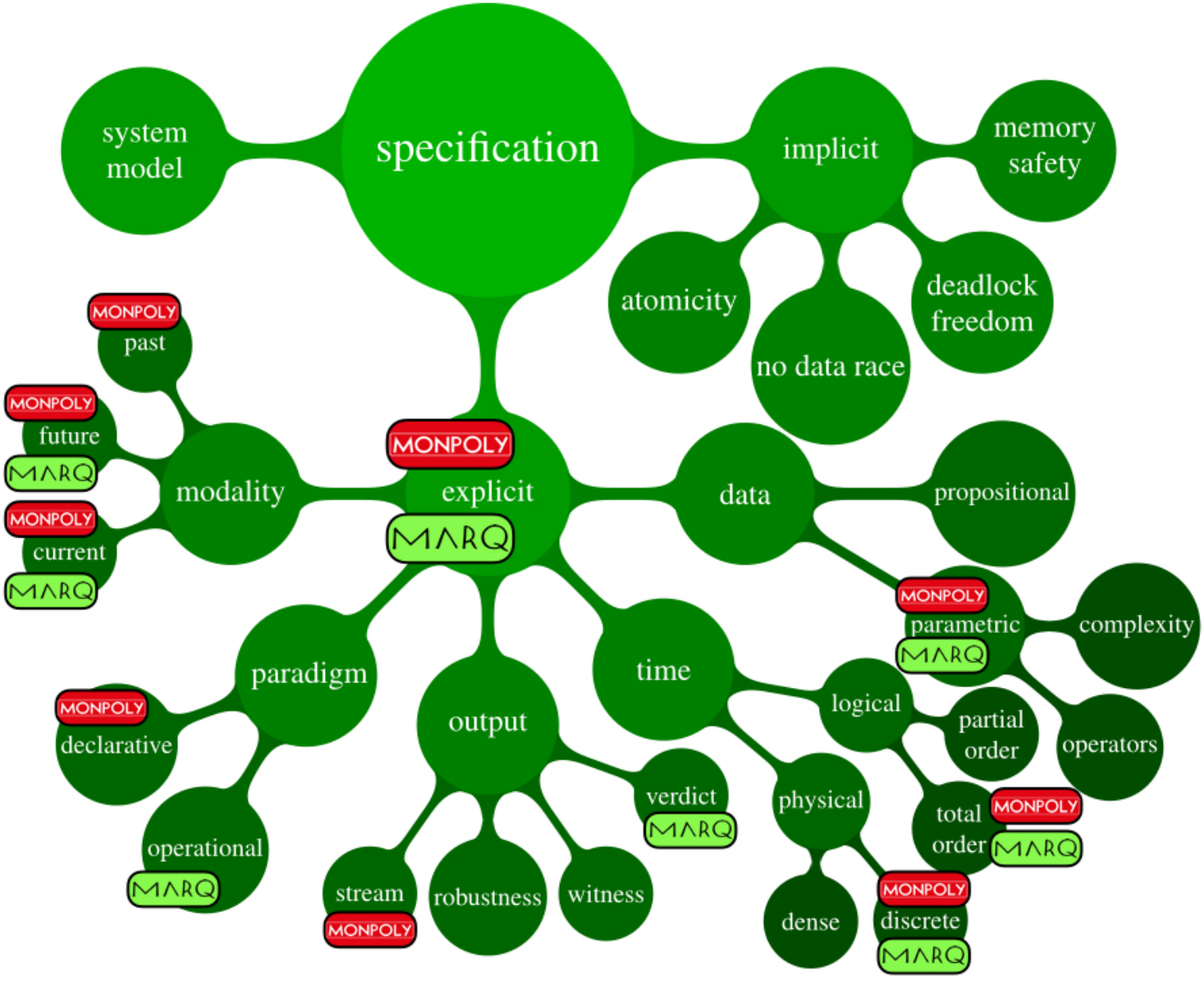
Specification



Specification



Specification



Monitor

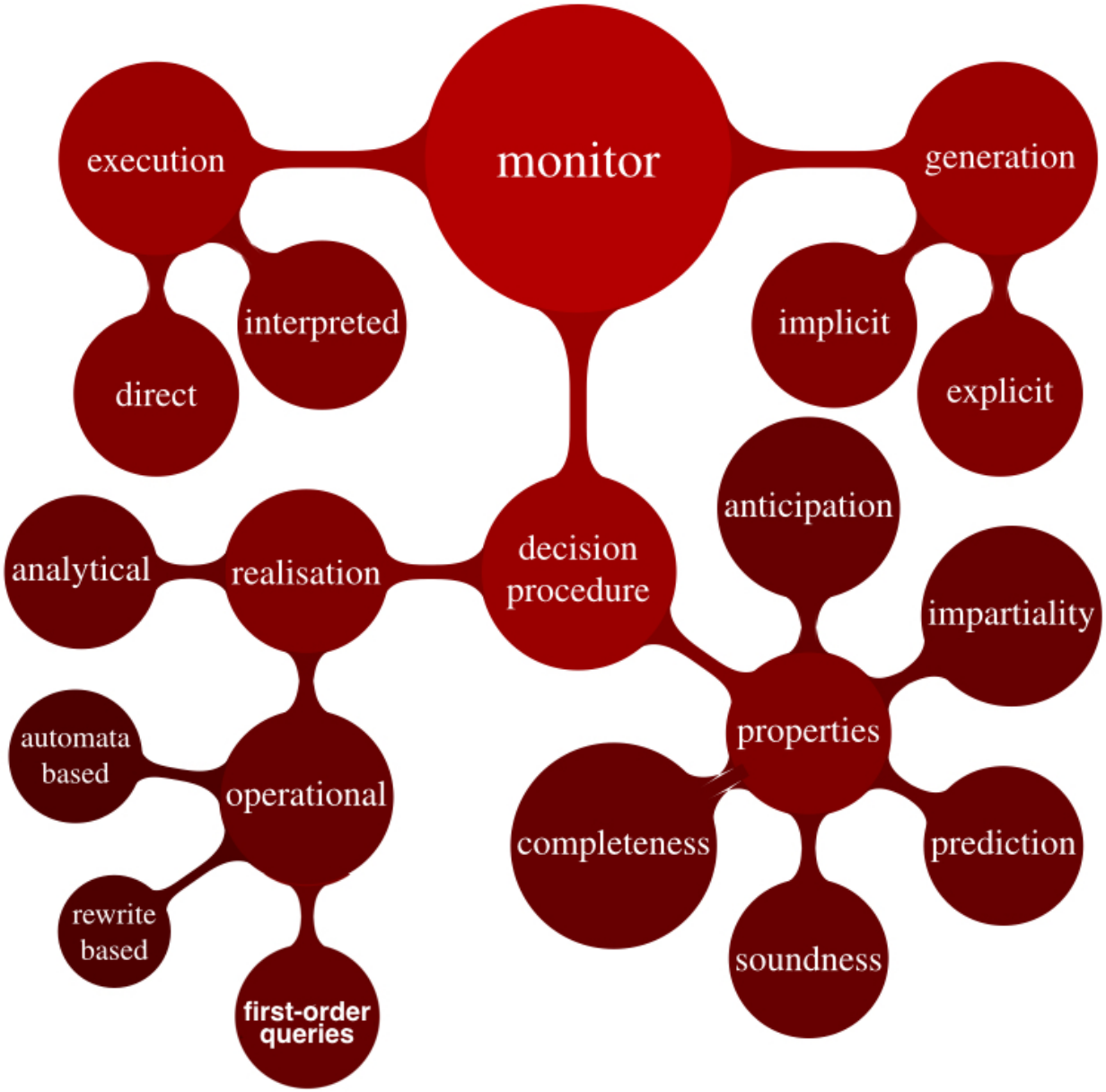
A monitor is a main component of a runtime verification framework.

In the previous sentence there was lots of discussion between a main and the main

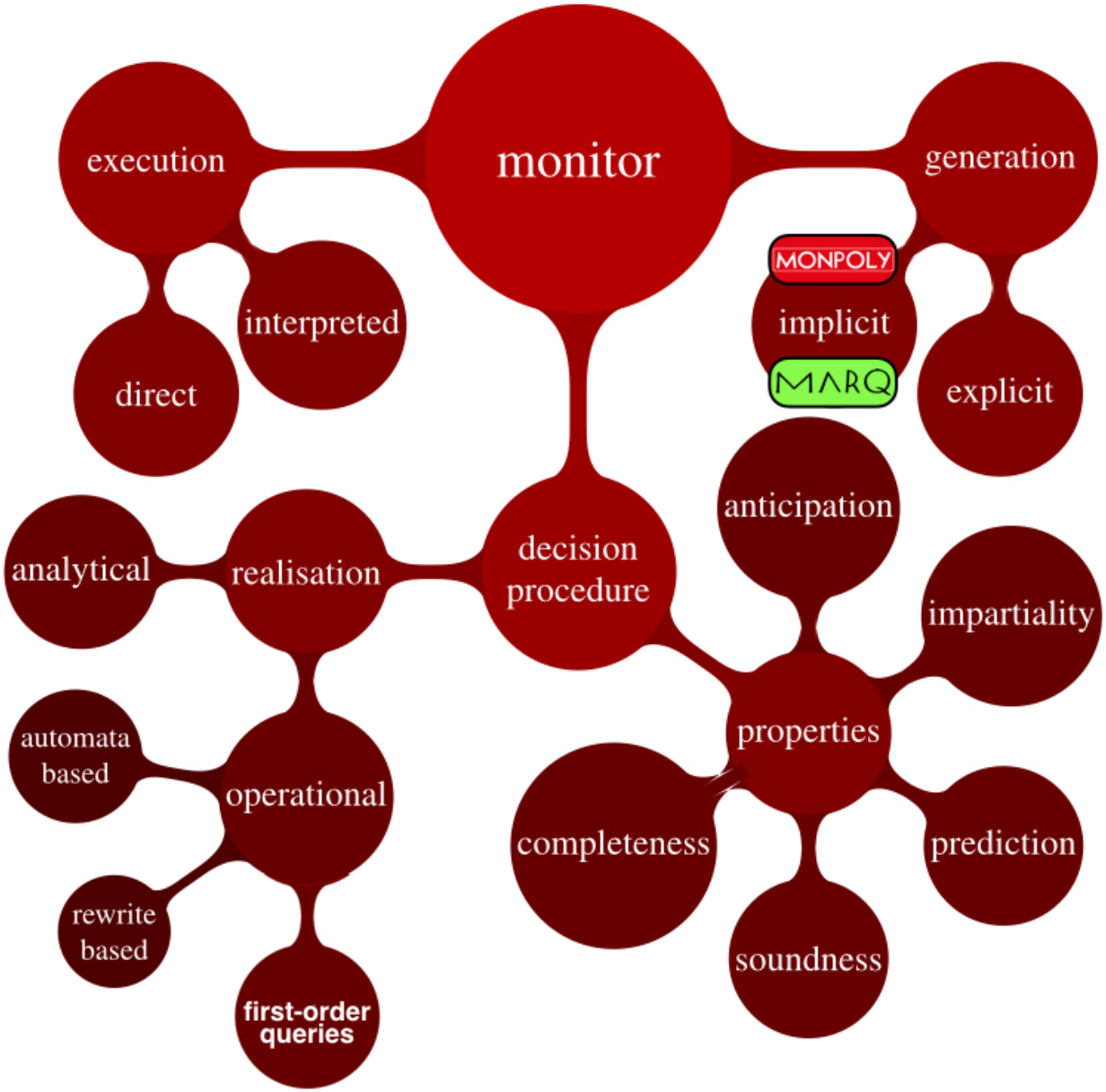
A monitor implements a decision procedure which produces the expected output (defined by the specification language)

At this point let us point out that our taxonomy does not (currently) capture non-hierarchical relationships between concepts

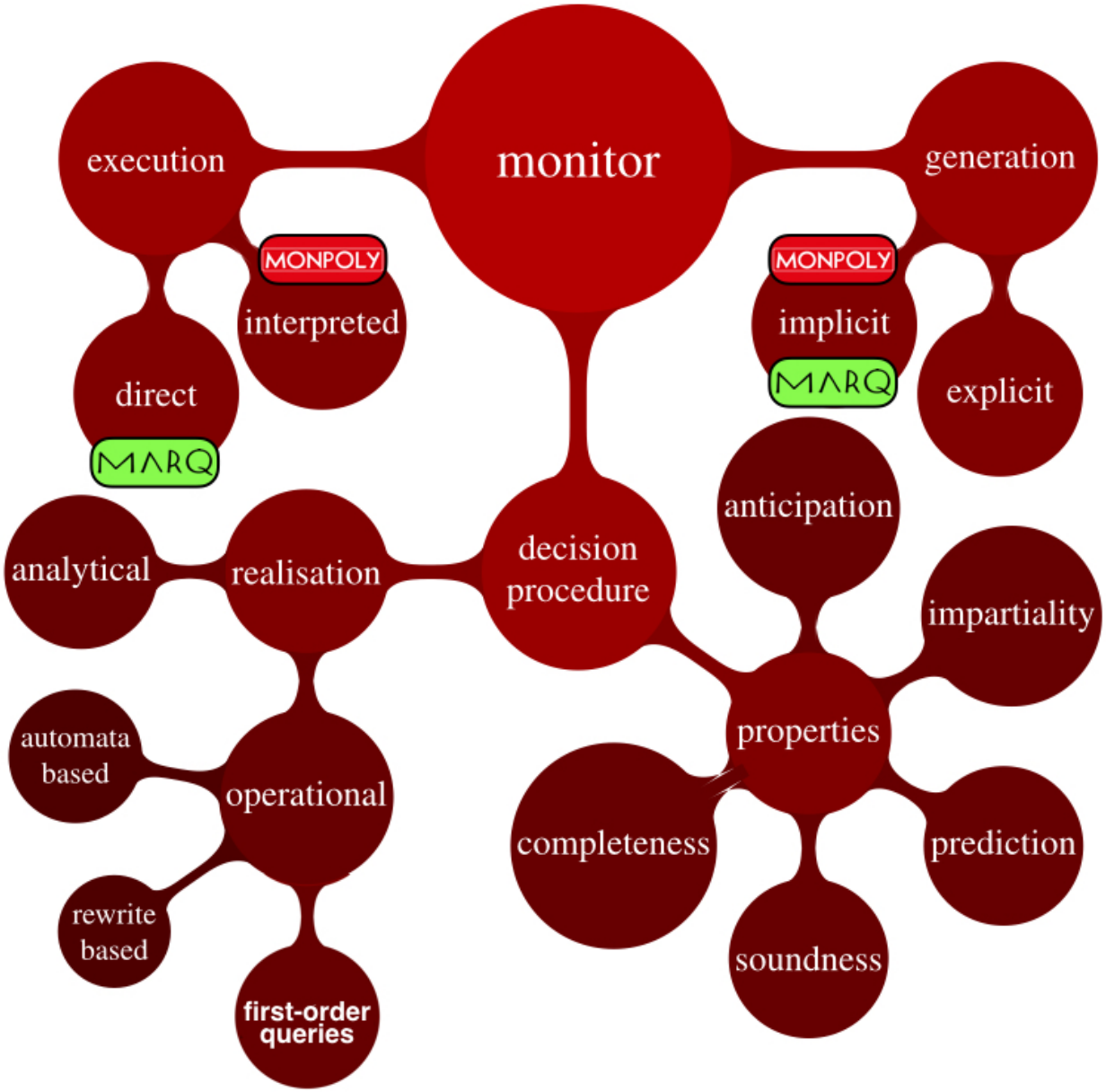
Monitor


monitor
execution
generation
interpreted
direct
implicit
explicit
anticipation
analytical
realisation
decision procedure
impartiality
automata based
operational
properties
completeness
prediction
rewrite based
soundness
first-order queries

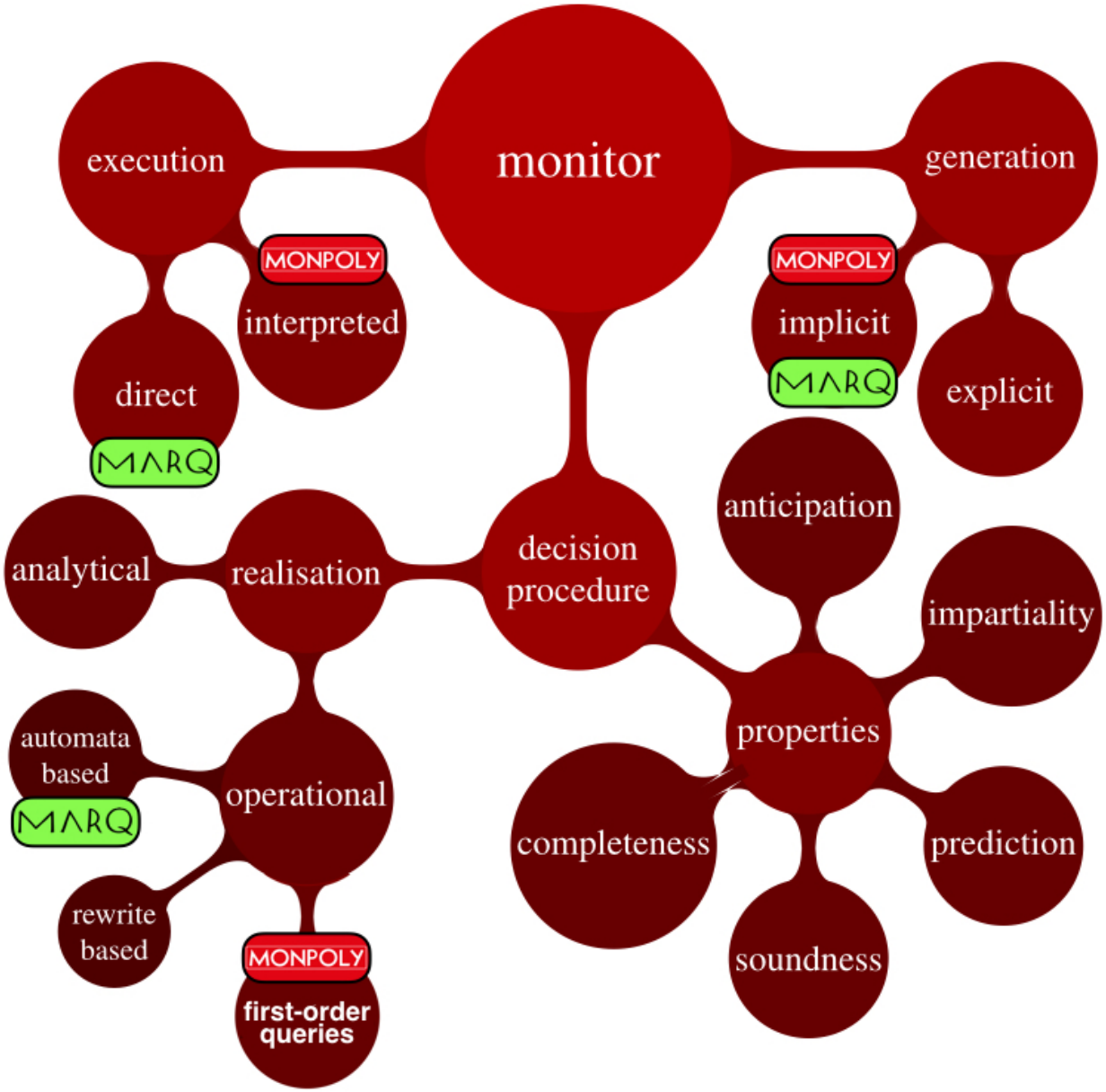
Monitor


monitor
execution
interpreted
direct
generation
MONPOLY
implicit
MARQ
explicit
decision procedure
analytical
realisation
automata based
operational
rewrite based
first-order queries
anticipation
impartiality
properties
completeness
prediction
soundness

Monitor


monitor
execution
generation
MONPOLY
interpreted
direct
MARQ
MONPOLY
implicit
MARQ
explicit
anticipation
impartiality
analytical
realisation
decision procedure
properties
automata based
operational
rewrite based
first-order queries
completeness
prediction
soundness

Monitor

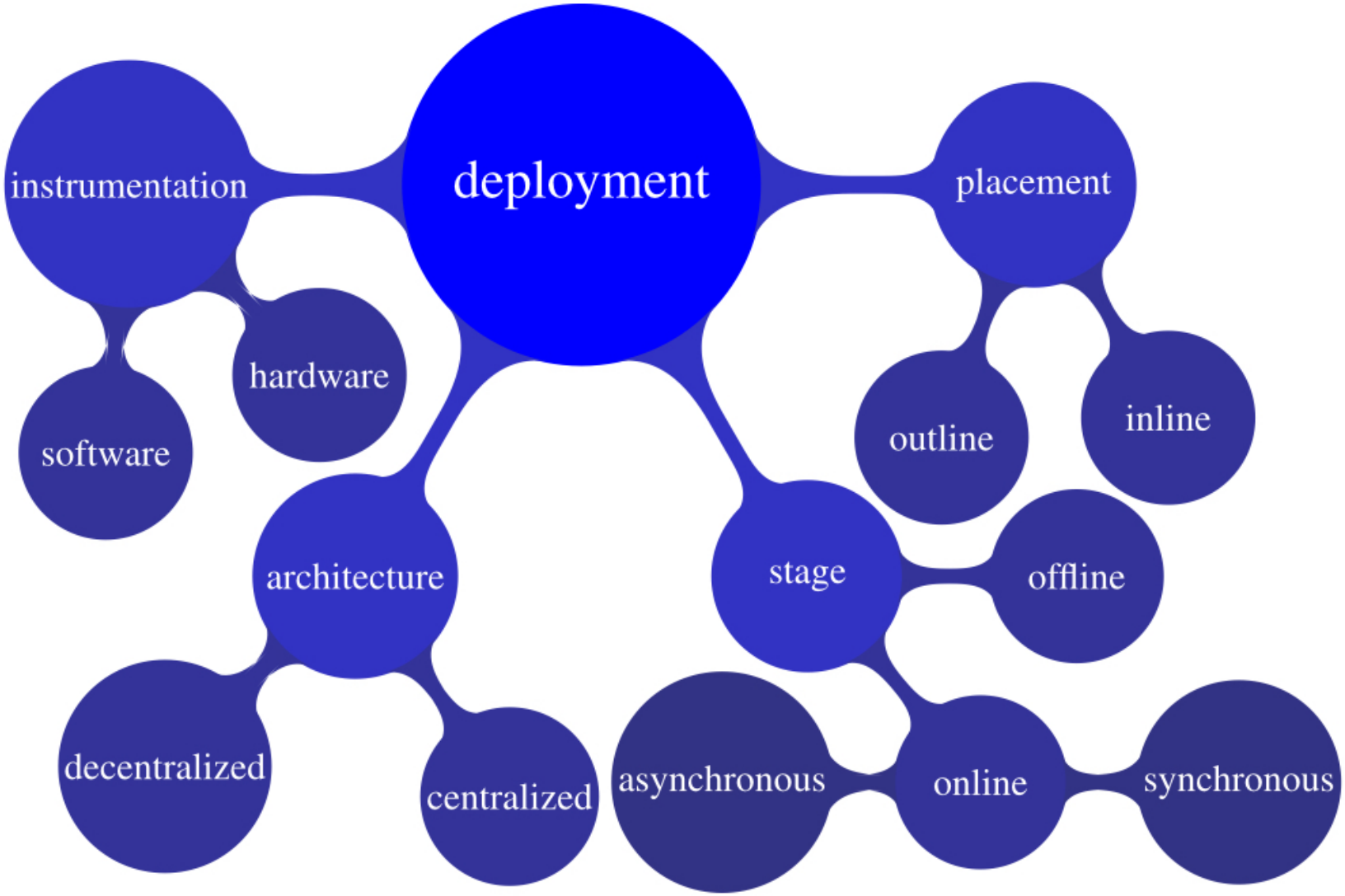

monitor
execution
generation
MONPOLY
interpreted
direct
MARQ
MONPOLY
implicit
MARQ
explicit
anticipation
impartiality
analytical
realisation
decision procedure
properties
automata based
MARQ
operational
completeness
prediction
rewrite based
MONPOLY
first-order queries
soundness

Deployment

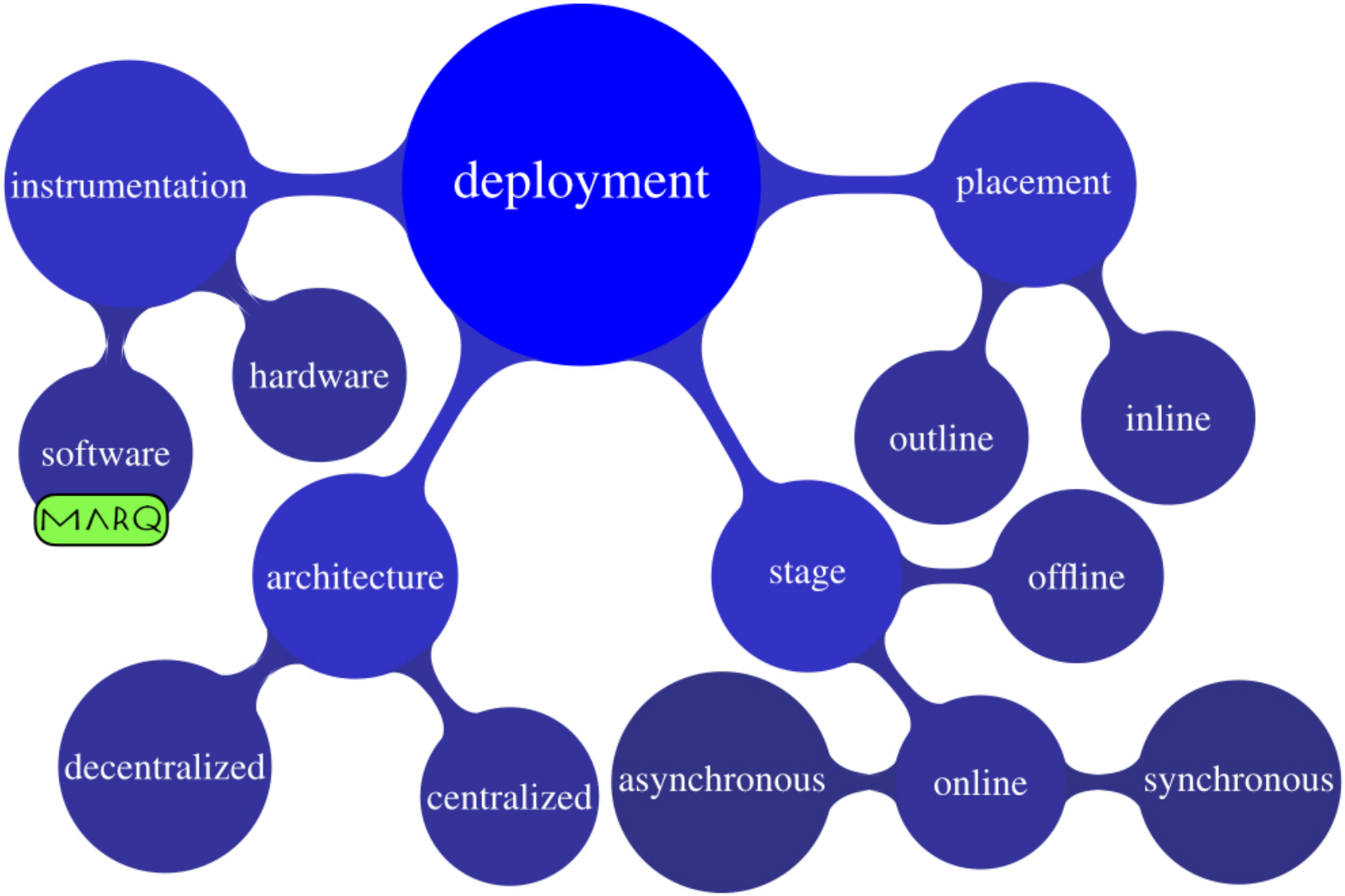
By deployment, we refer to how the monitor is effectively implemented, organized, how it retrieves the information from the system, and when it does so.

It became clear that there is not a general completely precise agreement between the exact meaning of online/offline and inline/outline - concepts that have been with us for over a decade.

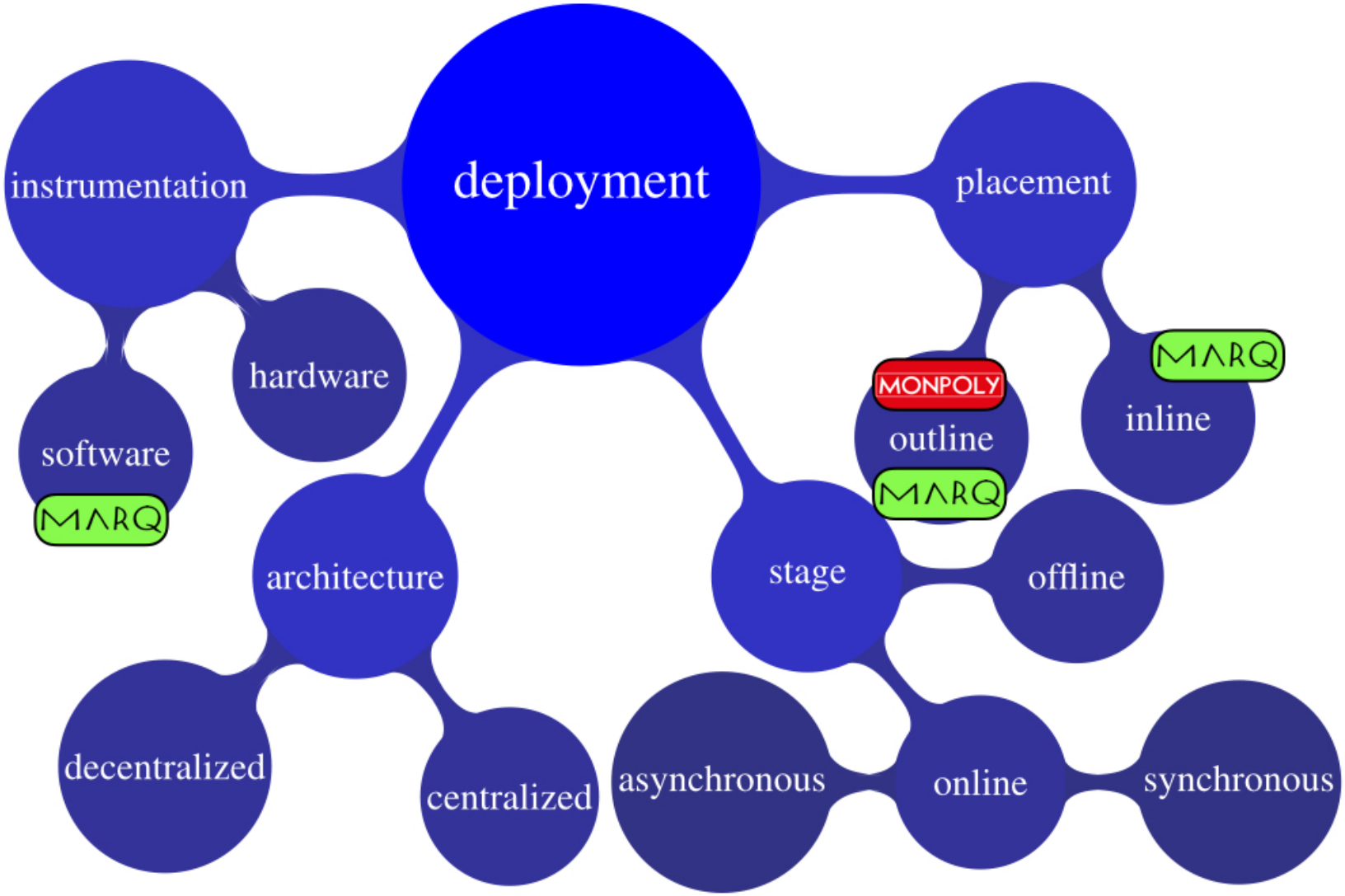
Deployment


deployment
instrumentation
hardware
software
placement
outline
inline
architecture
stage
offline
decentralized
centralized
asynchronous
online
synchronous

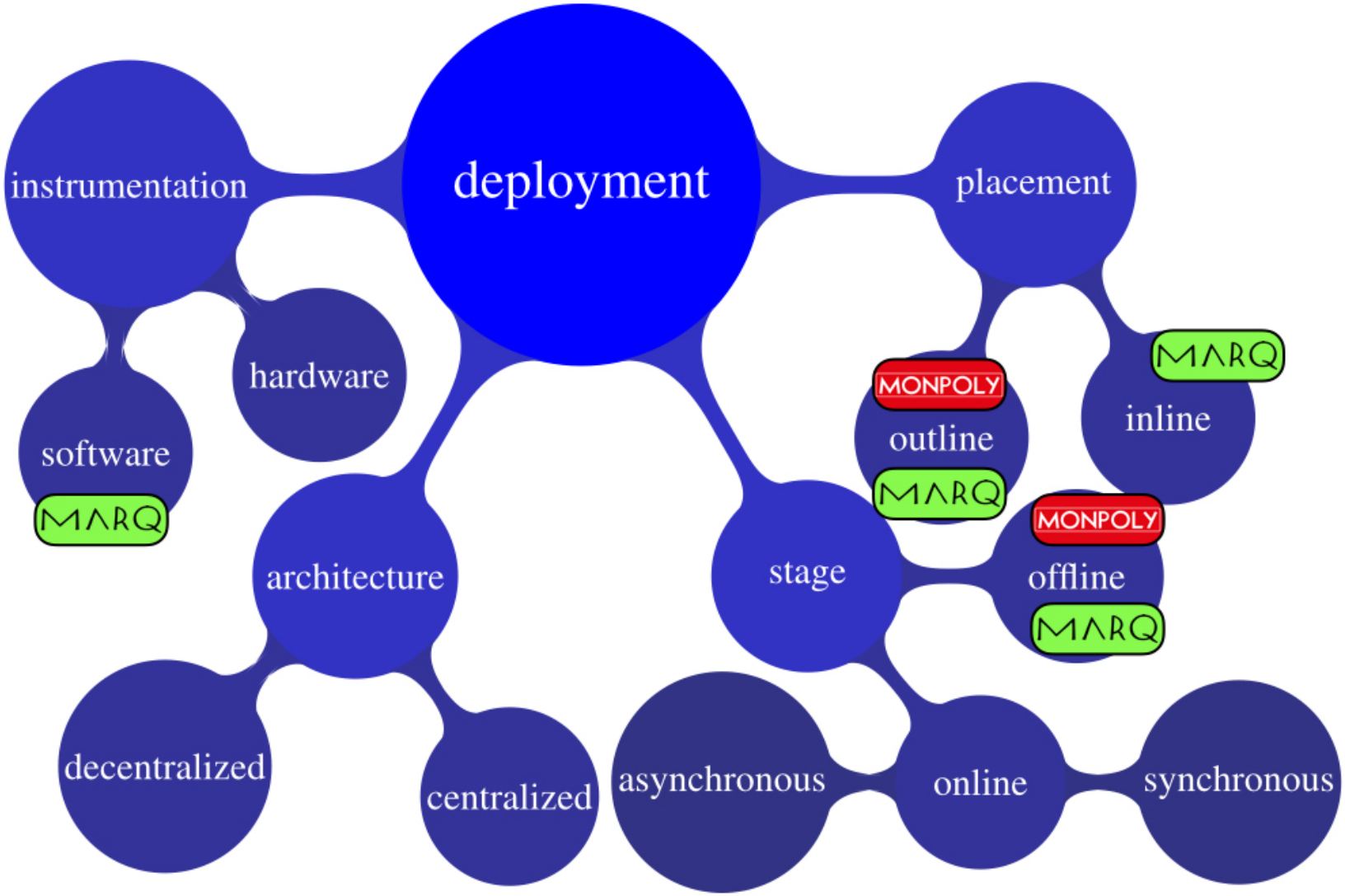
Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
outline
inline
offline
decentralized
centralized
asynchronous
online
synchronous

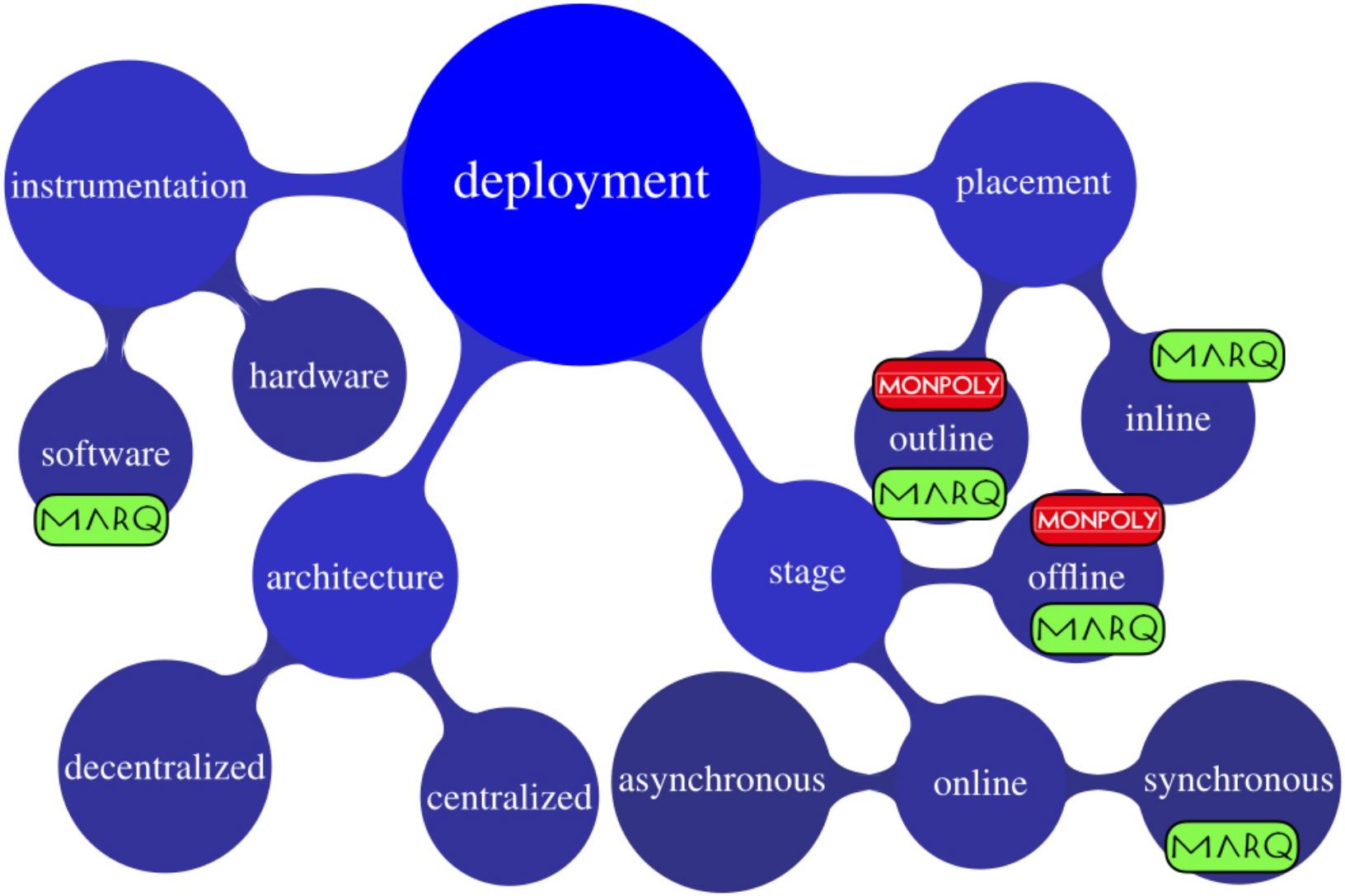
Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
MONPOLY
outline
MARQ
MARQ
inline
offline
decentralized
centralized
asynchronous
online
synchronous

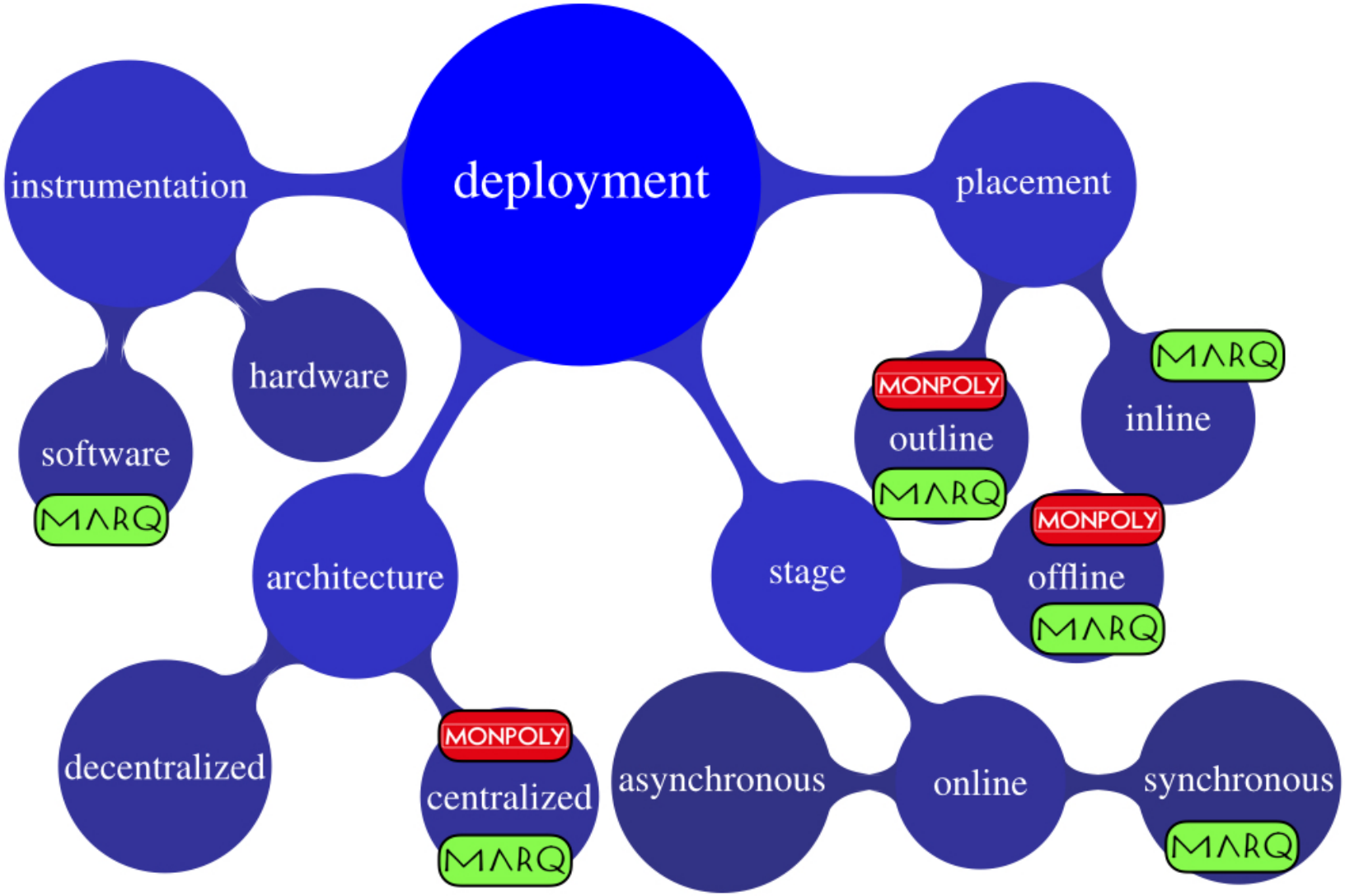
Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
MONPOLY
outline
MARQ
MARQ
inline
MONPOLY
offline
MARQ
decentralized
centralized
asynchronous
online
synchronous

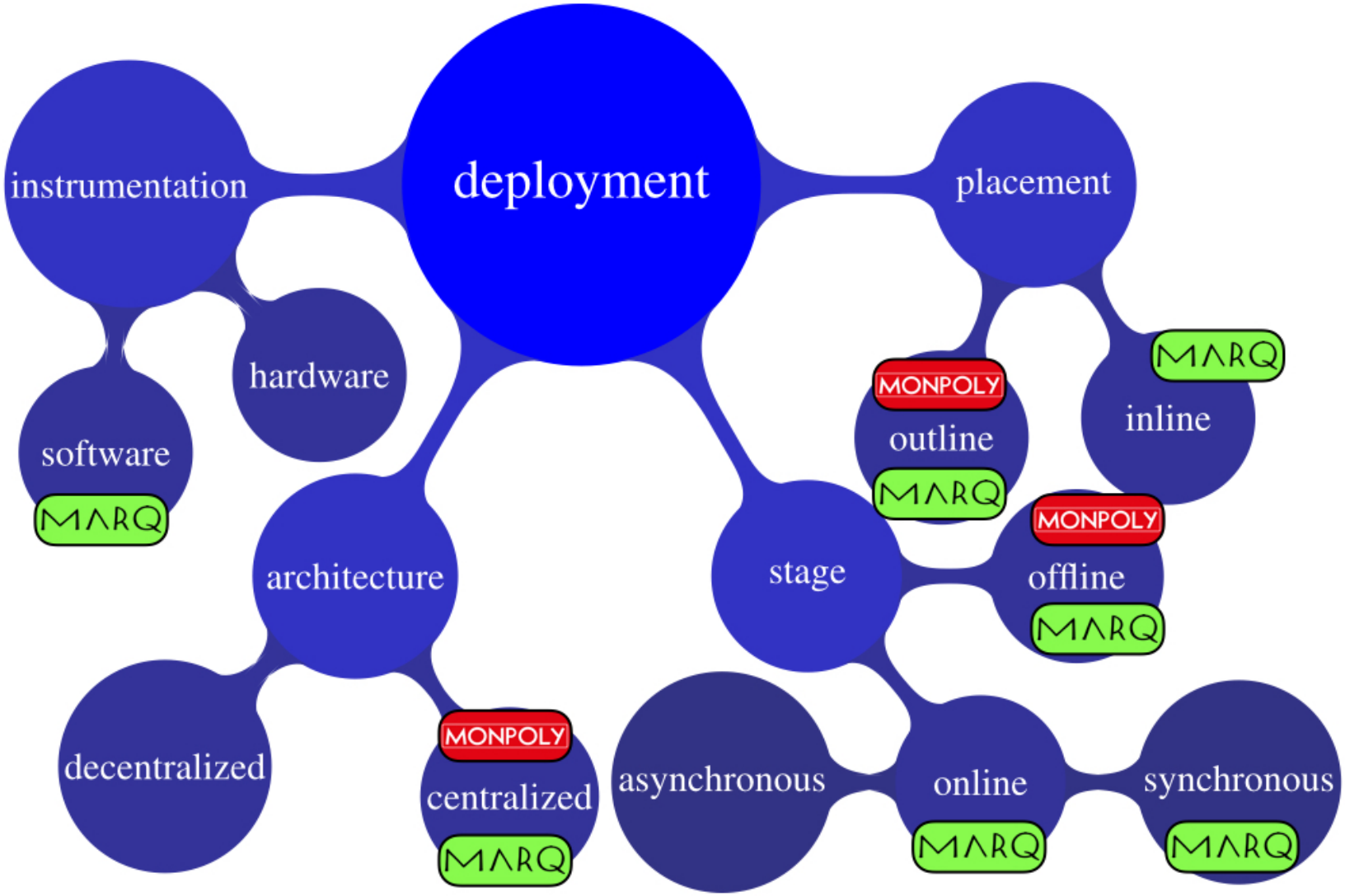
Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
MONPOLY
outline
MARQ
MARQ
inline
MONPOLY
offline
MARQ
decentralized
centralized
asynchronous
online
synchronous
MARQ

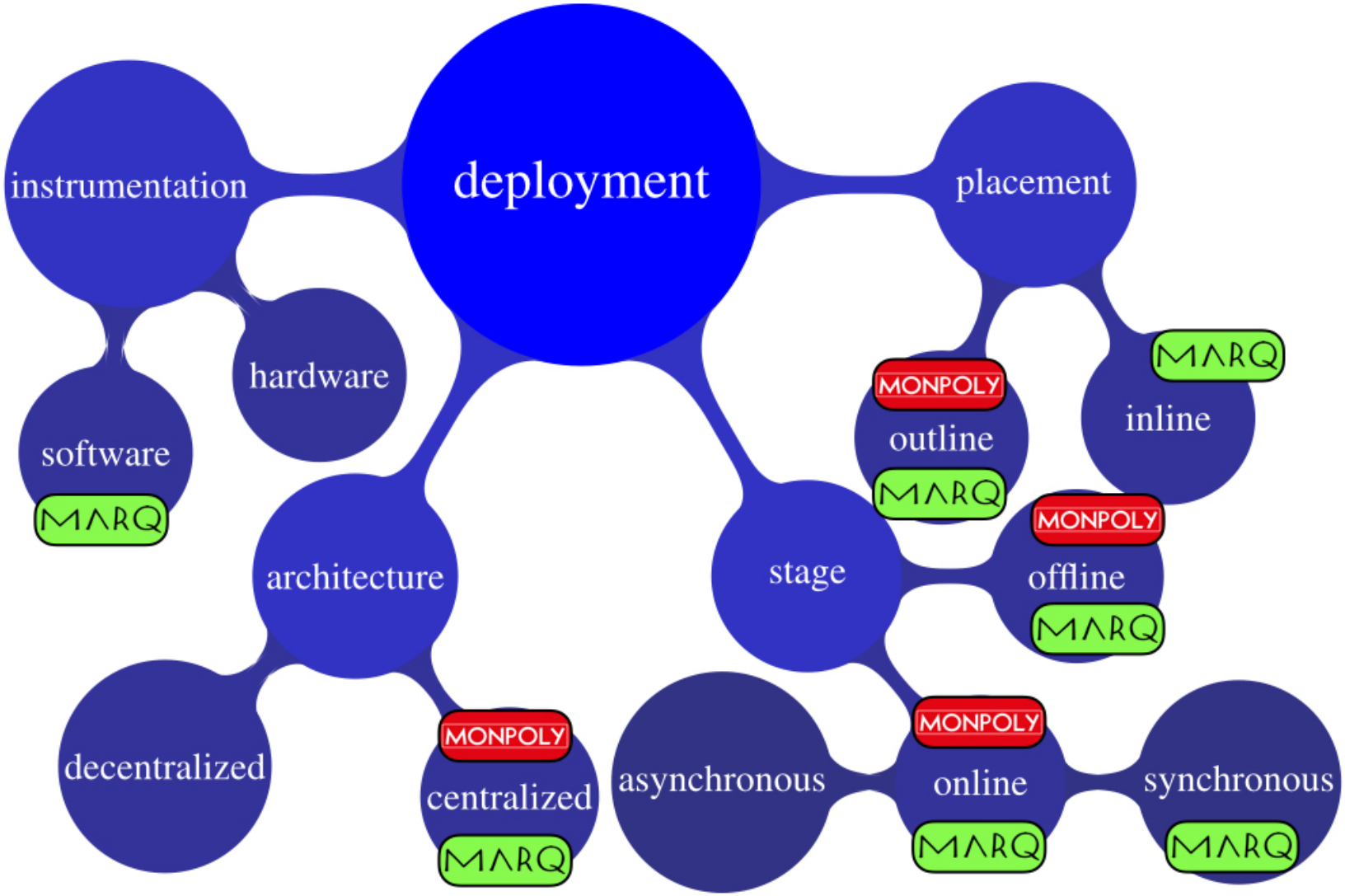
Deployment


deployment
instrumentation
software
MARQ
hardware
architecture
decentralized
MONPOLY
centralized
MARQ
placement
MONPOLY
outline
MARQ
MARQ
inline
stage
MONPOLY
offline
MARQ
asynchronous
online
synchronous
MARQ

Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
MONPOLY
outline
MARQ
MARQ
inline
MONPOLY
offline
MARQ
decentralized
MONPOLY
centralized
MARQ
asynchronous
online
MARQ
synchronous
MARQ

Deployment


deployment
instrumentation
placement
hardware
software
MARQ
architecture
stage
MONPOLY
outline
MARQ
MARQ
inline
MONPOLY
offline
MARQ
decentralized
MONPOLY
centralized
MARQ
asynchronous
MONPOLY
online
MARQ
synchronous
MARQ

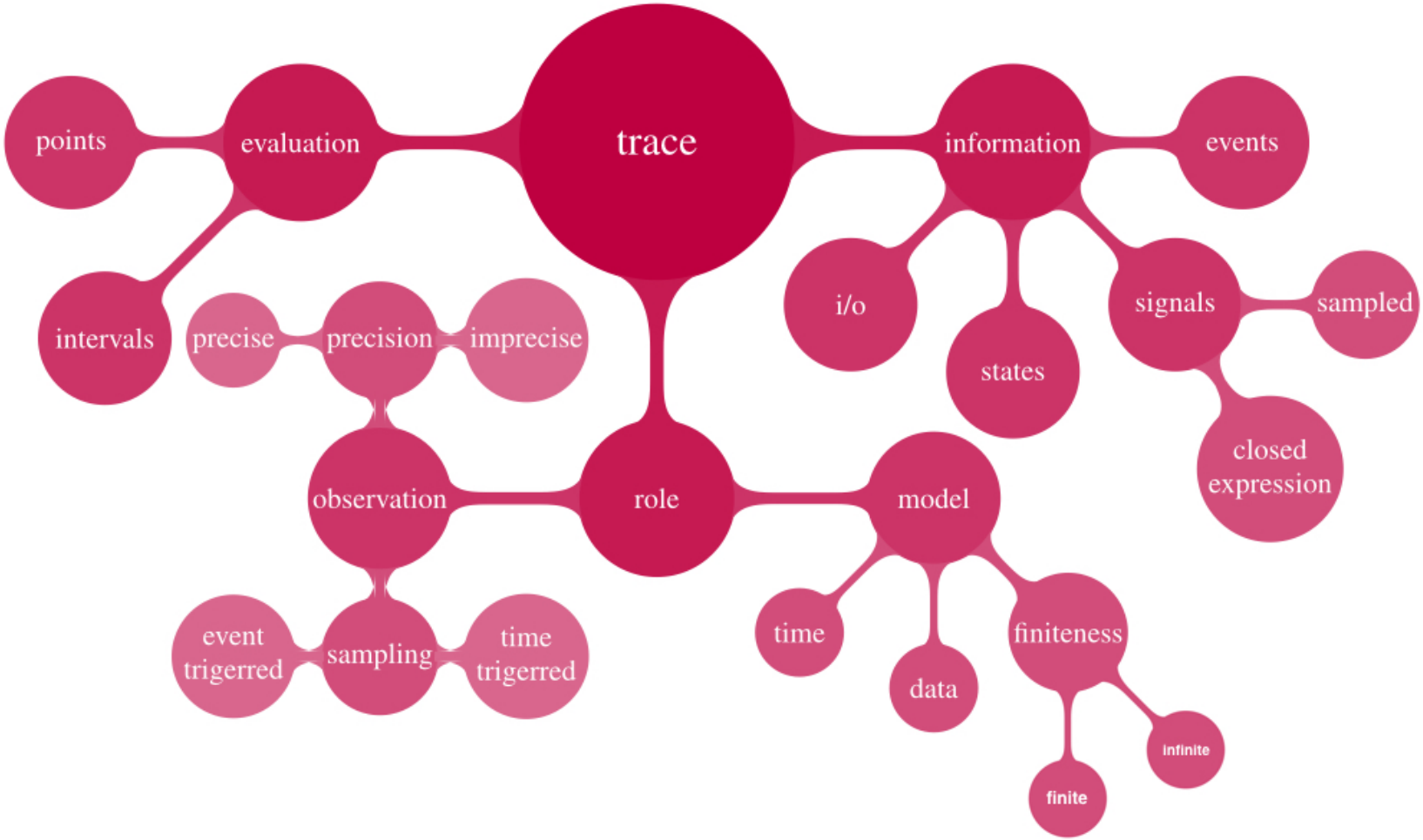
Trace

The notion of trace appears in two places in a runtime verification framework and this distinction is captured by the *role* concept.

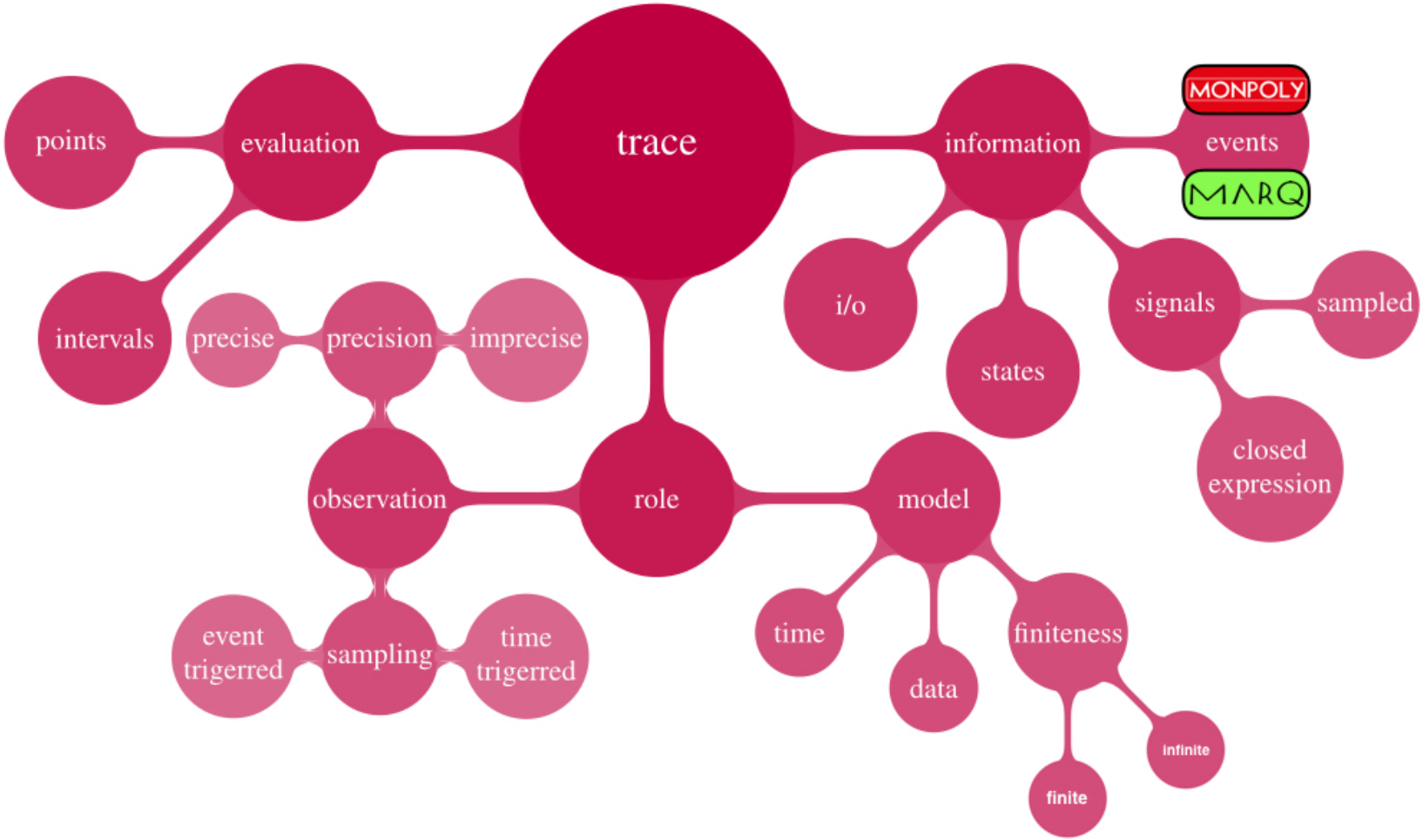
By *observed* trace we refer to the object extracted from the monitored system and examined by the monitor.

Conversely the trace *model* is the mathematical object forming part of the semantics of the specification formalism.

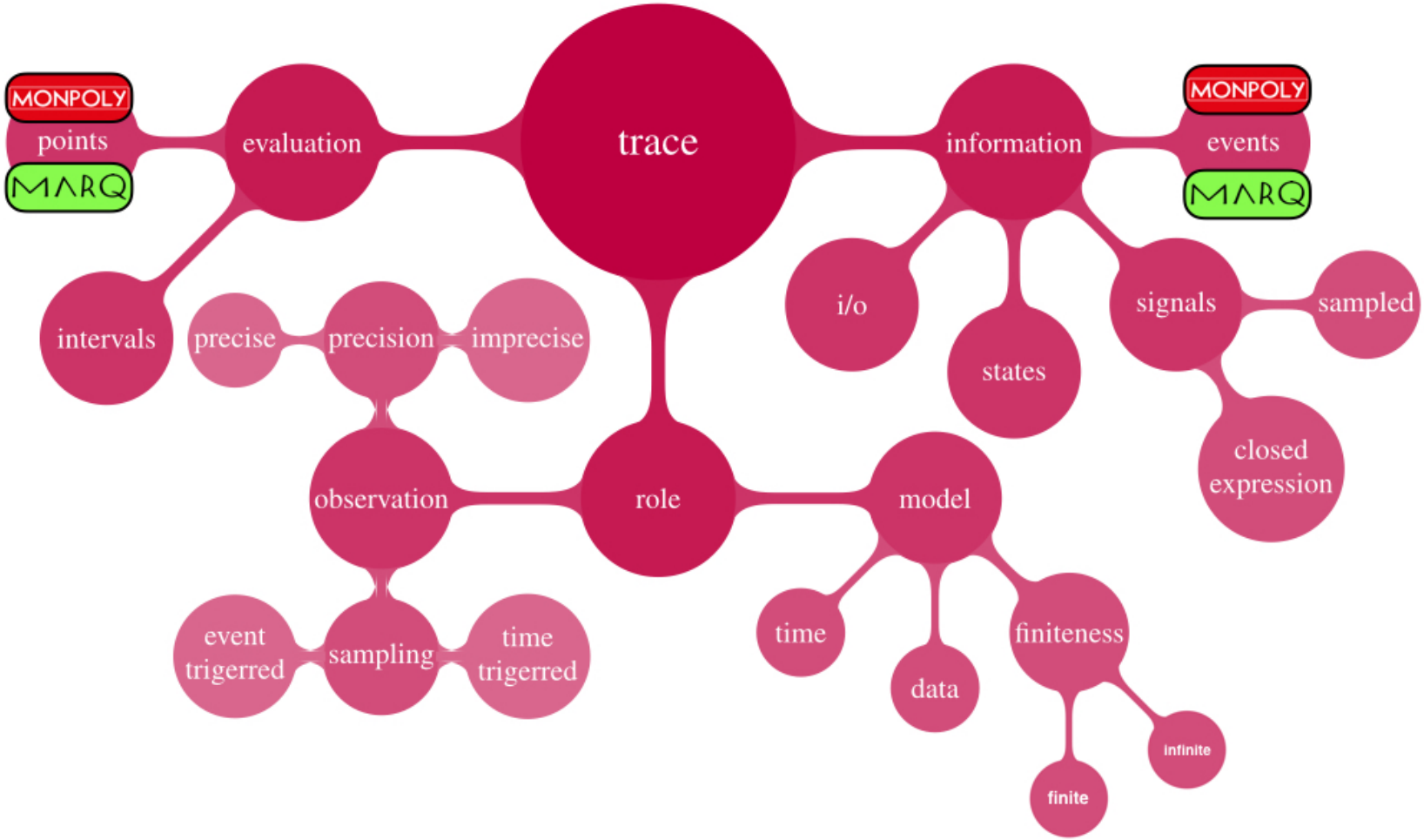
Trace


trace
evaluation
points
intervals
information
events
i/o
states
signals
sampled
closed expression
role
observation
precise
precision
imprecise
event trigerred
sampling
time trigerred
model
time
data
finiteness
infinite
finite

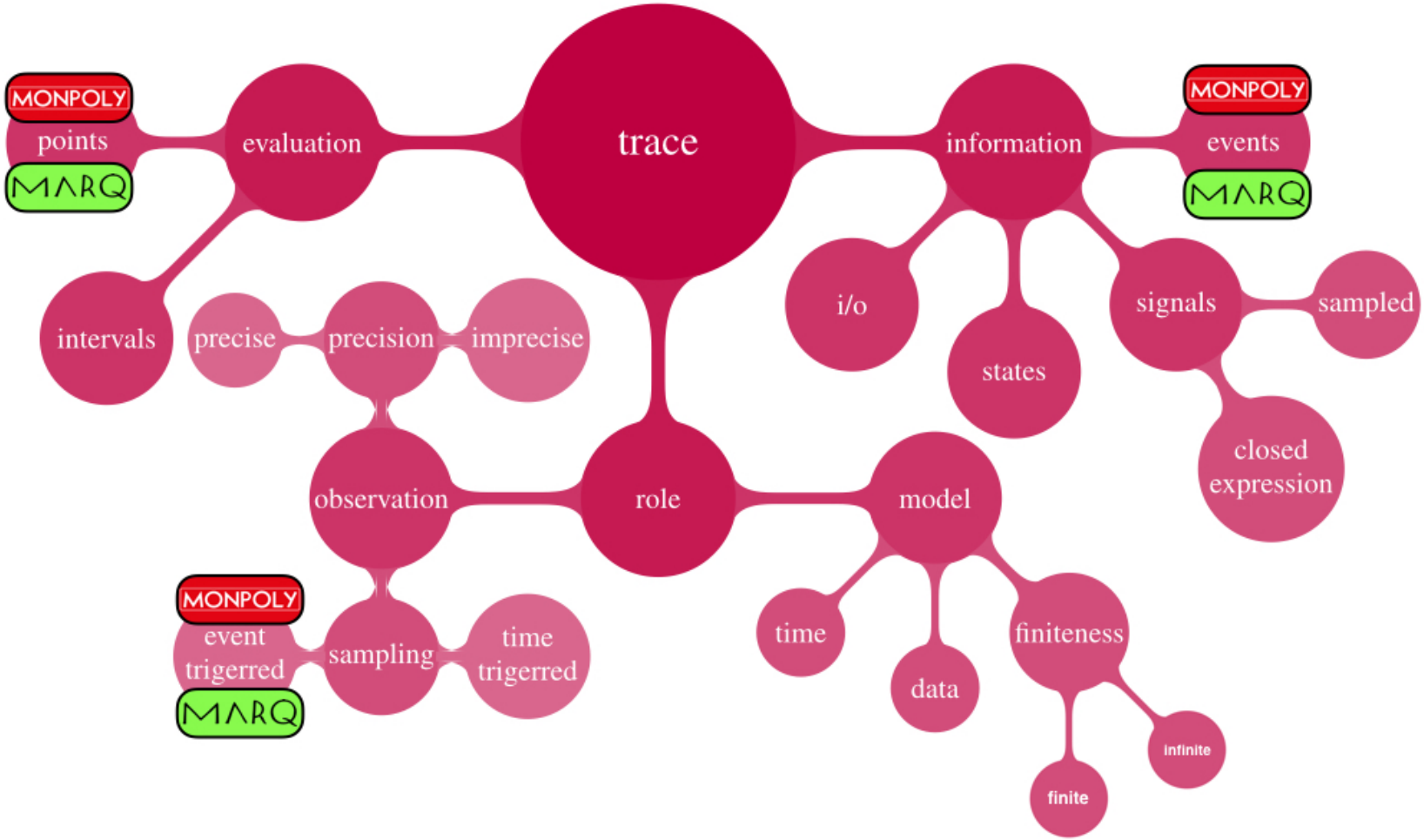
Trace


trace
evaluation
points
intervals
information
MONPOLY
events
MARQ
i/o
states
signals
sampled
closed expression
role
observation
precise
precision
imprecise
event trigerred
sampling
time trigerred
model
time
data
finiteness
infinite
finite

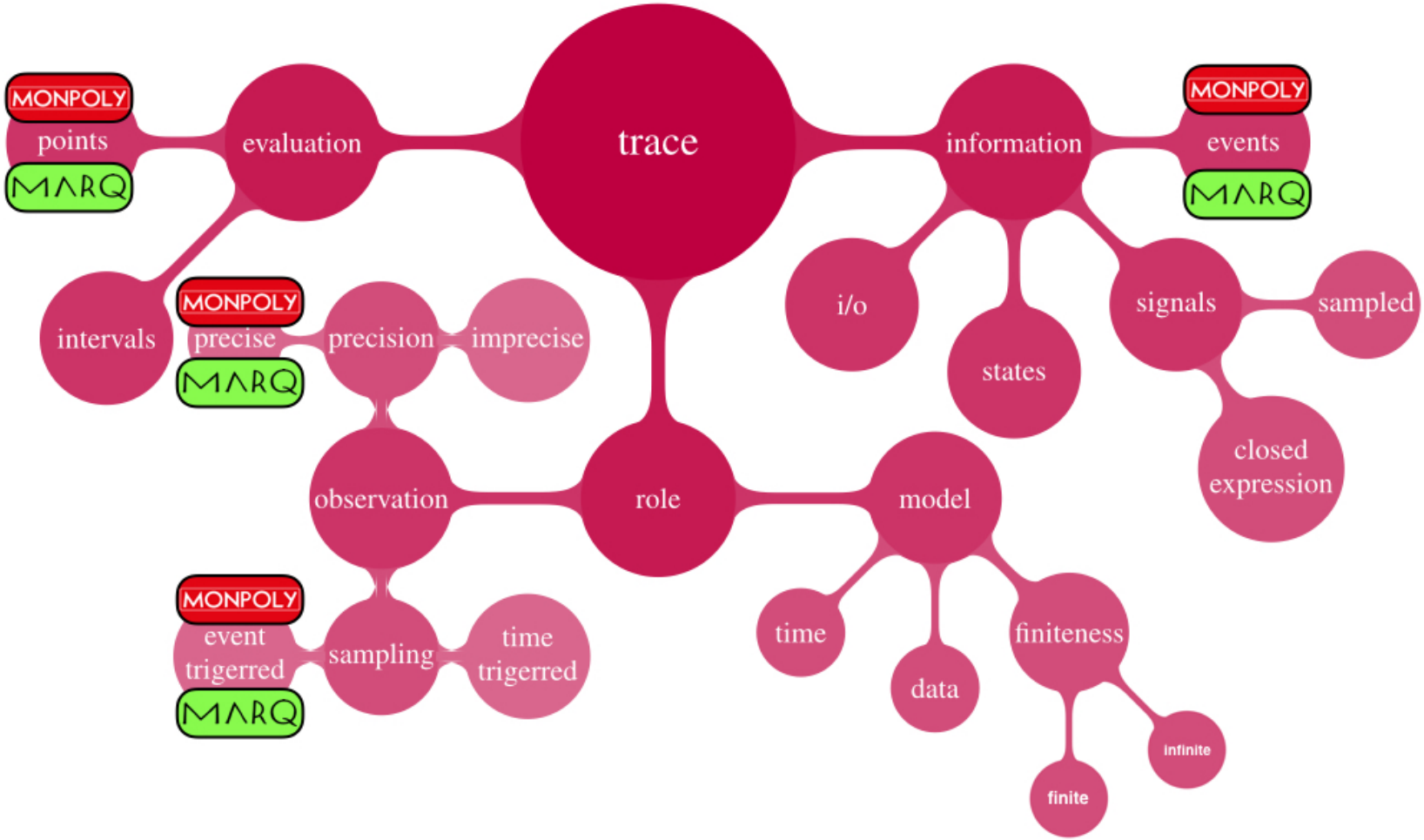
Trace


MONPOLY
points
MARQ
evaluation
trace
information
MONPOLY
events
MARQ
intervals
precise
precision
imprecise
i/o
states
signals
sampled
closed
expression
observation
role
model
event
trigerred
sampling
time
trigerred
time
data
finiteness
infinite
finite

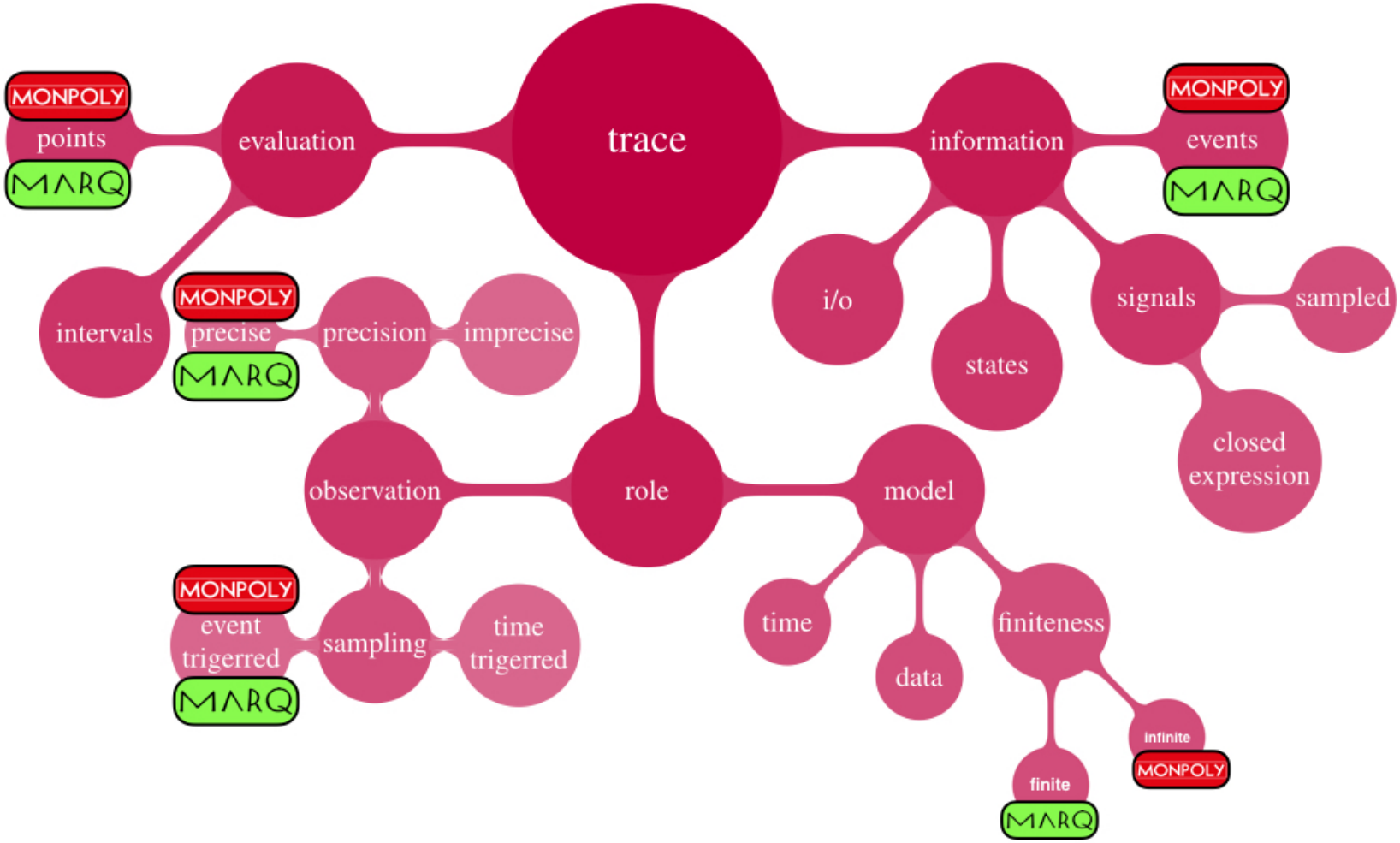
Trace


trace
evaluation
MONPOLY
points
MARQ
intervals
precise
precision
imprecise
observation
MONPOLY
event
trigerred
MARQ
sampling
time
trigerred
role
information
MONPOLY
events
MARQ
i/o
states
signals
sampled
closed
expression
model
time
data
finiteness
infinite
finite

Trace


trace
evaluation
MONPOLY
points
MARQ
intervals
MONPOLY
precise
MARQ
precision
imprecise
observation
MONPOLY
event
trigerred
MARQ
sampling
time
trigerred
role
information
MONPOLY
events
MARQ
i/o
states
signals
sampled
closed
expression
model
time
data
finiteness
infinite
finite

Trace


trace
evaluation
MONPOLY
points
MARQ
intervals
MONPOLY
precise
MARQ
precision
imprecise
observation
MONPOLY
event
trigerred
MARQ
sampling
time
trigerred
role
information
MONPOLY
events
MARQ
i/o
states
signals
sampled
closed
expression
model
time
data
finiteness
finite
MARQ
infinite
MONPOLY

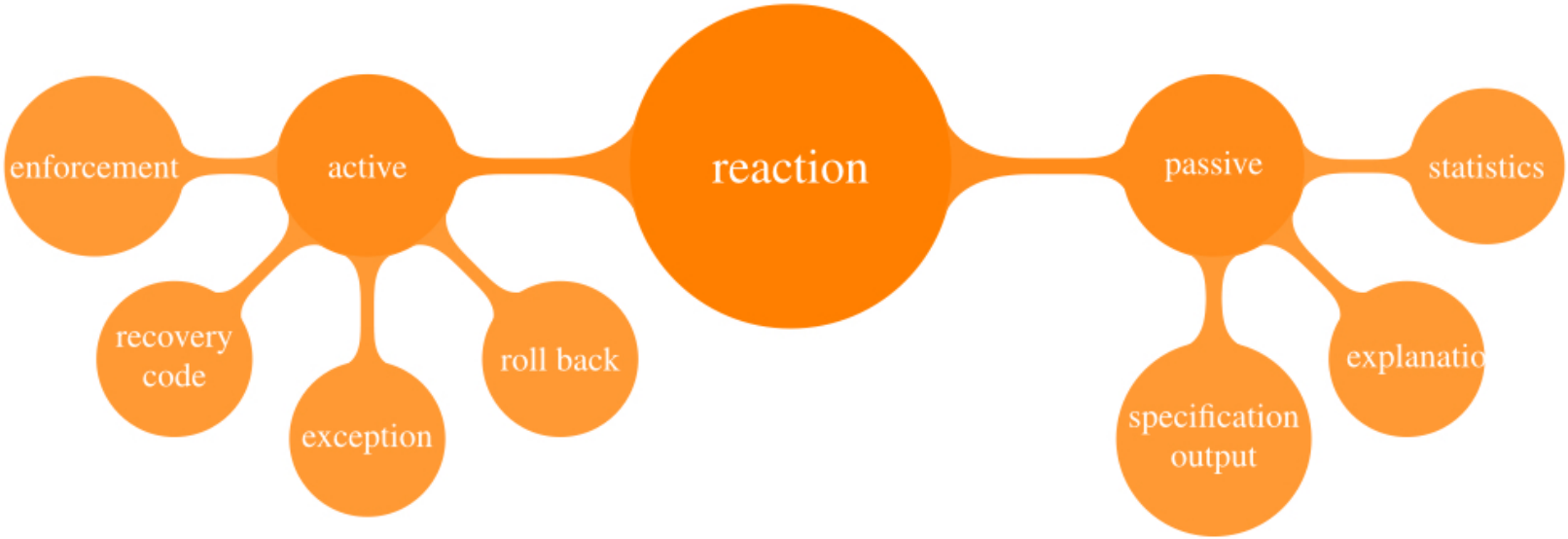
Reaction

By reaction, we refer to how the monitor affects the execution of the system; this can be passive or active.

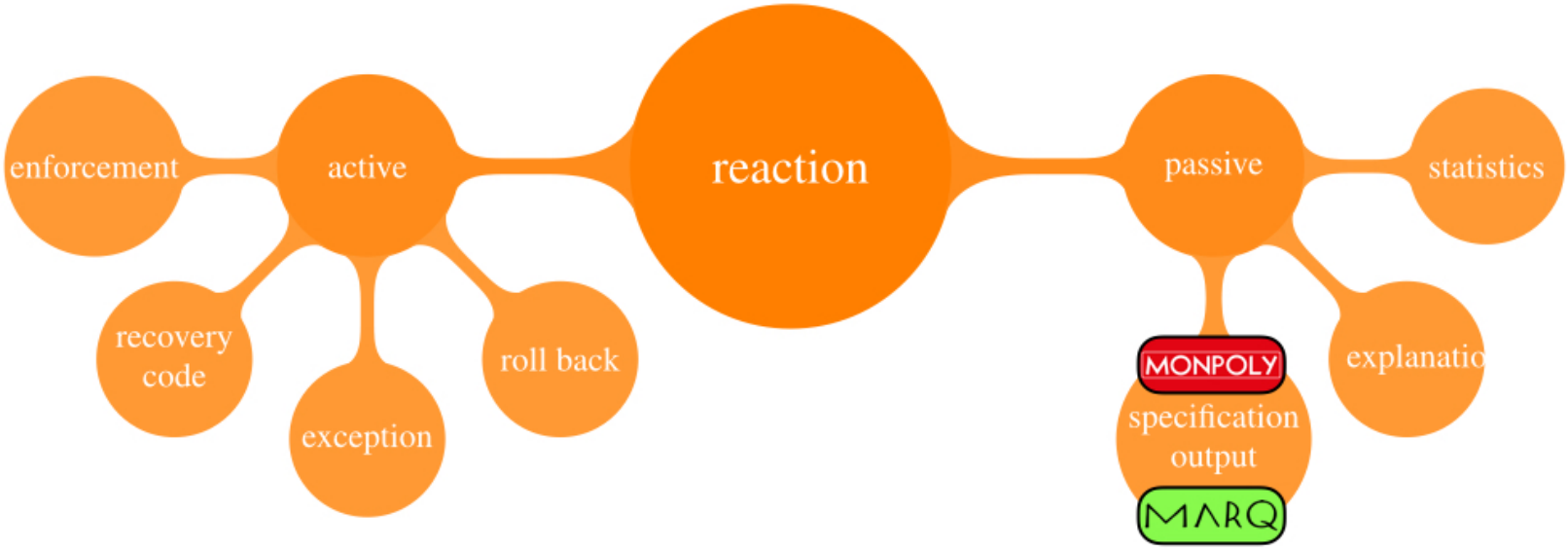
Reaction is said to be *passive* when the monitor does not influence or minimally influences the initial execution of the program.

Reaction is said to be *active* when the monitor affects the execution of the monitored system.

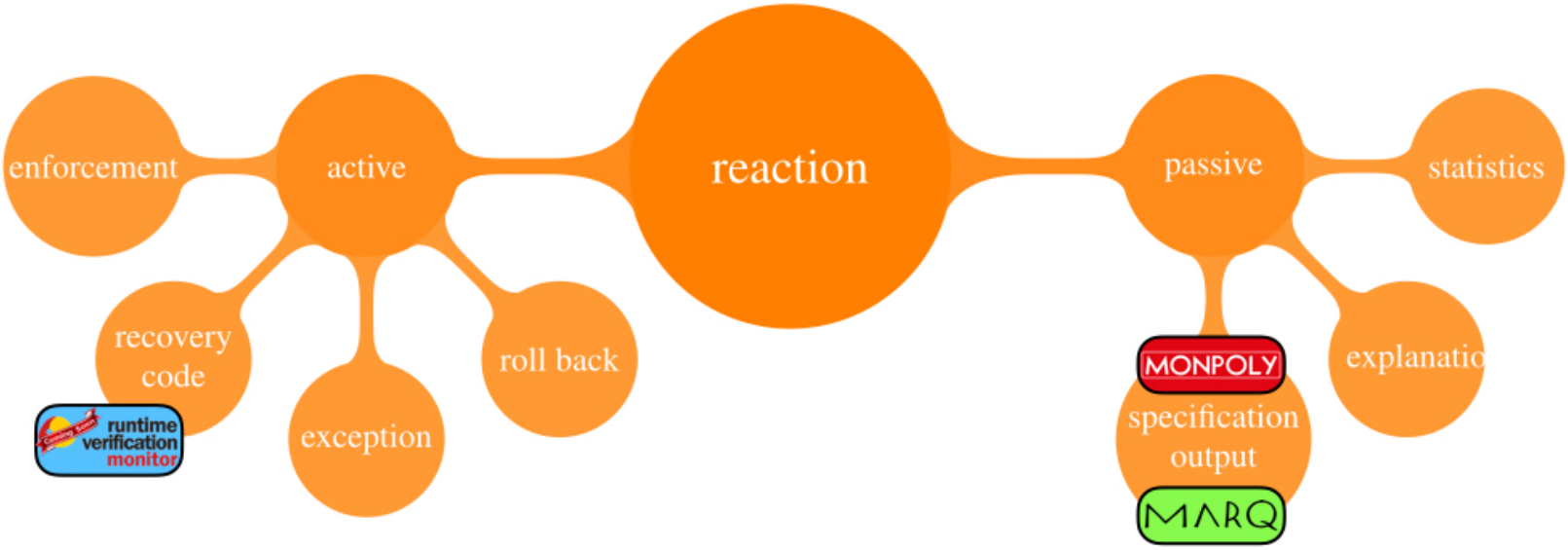
Reaction


enforcement
active
reaction
passive
statistics
recovery code
exception
roll back
specification output
explanatio

Reaction


enforcement
active
reaction
passive
statistics
recovery code
exception
roll back
MONPOLY
specification output
MARQ
explanatio

Reaction


reaction
enforcement
active
passive
statistics
recovery code
roll back
exception
explanatio
MONPOLY
specification output
MARQ
runtime verification monitor

Interference

he *interference* part of the taxonomy characterizes monitoring frameworks as *invasive* or *non-invasive*.

In absolute, a non-invasive monitoring framework being impossible (observer effect), this duality corresponds more in reality to a spectrum.

This part of the taxonomy is heavily related to other parts but we have not yet captured this or expanded this spectrum.

Application Area

Often an important part of what defines an RV tool.

Although some RV tools are developed to be agnostic there use has usually been within a set of domains.

Can heavily influence other parts e.g. kinds of properties, deployment restrictions, instrumentation techniques.

We do not provide a list of possible domains but do give some examples.

Classification (Tools)

Tools were taken from previous RV competitions and RV-CuBES workshop. This led to 20 tools in total.

Aerial	E-ACSL	MarQ	RTC
ARTiMon	JavaMOP	MonPoly	RV-Monitor
BeepBeep	jUnitRV	Mufin	STePr
DANA	Larva	R2U2	TemPsy/OCLR-Check
detectEr	LogFire	RiTHM	VALOUR

Threat to Validity. The competition was relatively restricted in the kinds of tools that it targeted and, whilst RV-CuBES was open, this alone means that our selection is biased.

Classification (Results)

Tool	Specification							Monitor			Deployment					Reaction		Trace				
	implicit	explicit				modality	paradigm	decision procedure	generation	execution	stage	synchronisation	architecture	placement	instrumentation	active	passive	information	sampling	evaluation	precision	model
		data	output	time: logical	time: physical																	
Aerial	none	p	s	tot	$\mathbb{N}$	all	d	dynamic programming	i	i	on	none	c	out	none	none	so	e	et	p	p	i
ARTiMon	none	s	s	tot	$\mathbb{NR}$	all	d	?	i	i	on	none	c	out	none	none	so	e	et	i	p	i
BeepBeep	none	c	s	tot	none	f	all	stream-processing	i	d	on	all	c	out	sw	none	so	e	et	p	p	i
DANA	none	p	s	tot	$\mathbb{R}$	all	o	?	i	d	on	sync	c	?	?	none	so	e	et	p	p	f
detectEr	none	s	v	par	none	f	d	dynamic programming	e	i	on	all	c	in	sw	none	so	e	et	p	p	i
E-ACSL	ms	na	r	?	na	na	o	code rewriting with assertions	e	d	on	sync	c	in	sw	e	e	s	na	na	na	na
JavaMOP	none	s	w	tot	none	all	all	trace slicing plugin-based	e	d	on	sync	c	in	swAJ	r	e	e	et	p	p	f
jUnitRV	none	s	v	tot	none	f	d	automata-based (modulo theories eg. SMT solver)	e	d	on	sync	c	in	swR	?	?	e	et	p	p	f
Larva	none	s	v	tot	$\mathbb{N}$	f	o	automata-based	e	d	on	all	c	all	sw	r	so	e	et	p	p	f
LogFire	none	s	w	tot	none	all	o	rewriting-based (RETE)	i	d	all	sync	c	out	none	none	so	e	et	p	p	f
MarQ/ QEA	none	s	v	tot	$\mathbb{N}$	f	o	automata-based	i	d	all	sync	c	all	sw	none	so	e	et	p	p	f
MonPoly	none	s	s	tot	$\mathbb{N}$	all	d	first-order queries	i	i	on	none	c	out	none	none	so	e	et	p	p	all
Mufin	none	s	v	tot	none	f	o	automata-based (union-find)	i	d	on	sync	c	out	none	none	so	e	et	p	p	f
R2U2	none	p	s	tot	$\mathbb{N}$	all	d	automata-based	e	i	on	async	c	out	none	none	so	e	et	p	p	i
RiTHM	none	p	s	tot	none	f	o	time-triggered runtime verification	e	d	on	async	c	in	sw	none	so	s	all	p	p	i
RTC	ms	na	w	?	na	na	o	?	i	d	on	sync	c	in	sw	r	?	?	et	p	p	na
RV-Monitor	none	s	w	tot	$\mathbb{N}$	all	all	(see JavaMOP)	i	d	all	sync	c	all	sw	r	e	e	et	p	p	f
STePr	none	s	s	tot	$\mathbb{N}$	all	o	?	i	d	on	?	c	out	none	none	so	e	et	p	p	?
TemPsy/ OCLR-Check	none	p	v	tot	$\mathbb{N}$	all	d	OCL constraint	i	i	off	na	c	out	none	none	so	e	et	p	p	f
VALOUR	none	s	v	tot	$\mathbb{N}$	all	o	automata-based	i	d	on	all	c	in	swAJ	none	all	e	all	p	p	f

Observations

Common themes

- Generally use totally ordered logical time
- Generally event-based & event-triggered
- Generally online
- Even split between ‘declarative’ and ‘operational’ languages

Most controversial part of taxonomy was Monitor

Most difficult to justify was relation between trace model and observed trace

Underpopulated Areas

Decentralized architecture - growing area (?)

Monitoring ‘states’ (instead of events) - common dichotomy (?)
not seen in tools (what does it mean to monitor state)

Richer reactions - mostly passive reactions, or very weak active ones

Applications - many tools were in fact application agnostic

Future Work

Tools

- Have around 30 more ready to classify
- Need to consult on current classification
- Main challenge: getting enough information
- Second challenge: some tools may require refinement of taxonomy

Refine Some Areas of Taxonomy

- Hardware monitoring
- Implicit property tools
- Currently only focus on ‘trace checking’ part of RV

Watch this Space

Survey

- In the next few months we will prepare a detailed survey to capture sufficient information to classify tools using our taxonomy.
- This will be distributed widely and we hope that you will help us by completing it.
- This will help us verify the current classification and significantly extend it.

Online Home

- To support the use of the taxonomy we are looking for an online home where the taxonomy and classification can easily be explored

Summary

First steps but big steps

Lots more work to do

Will other people use the taxonomy?

Will other people care about and/or contribute to the classification?