D-Wave
The Quantum Computing Company™

Tools Requirements and Progress Delivering to Them

Steve Reinhardt, Mike Booth, Alex Condello, Denny Dahl, Adam Douglass, and Murray Thom

Agenda

- Tools goals
- Tools requirements
- Deliveries to requirements since last Qubits
- Qubits 2022 tools capabilities

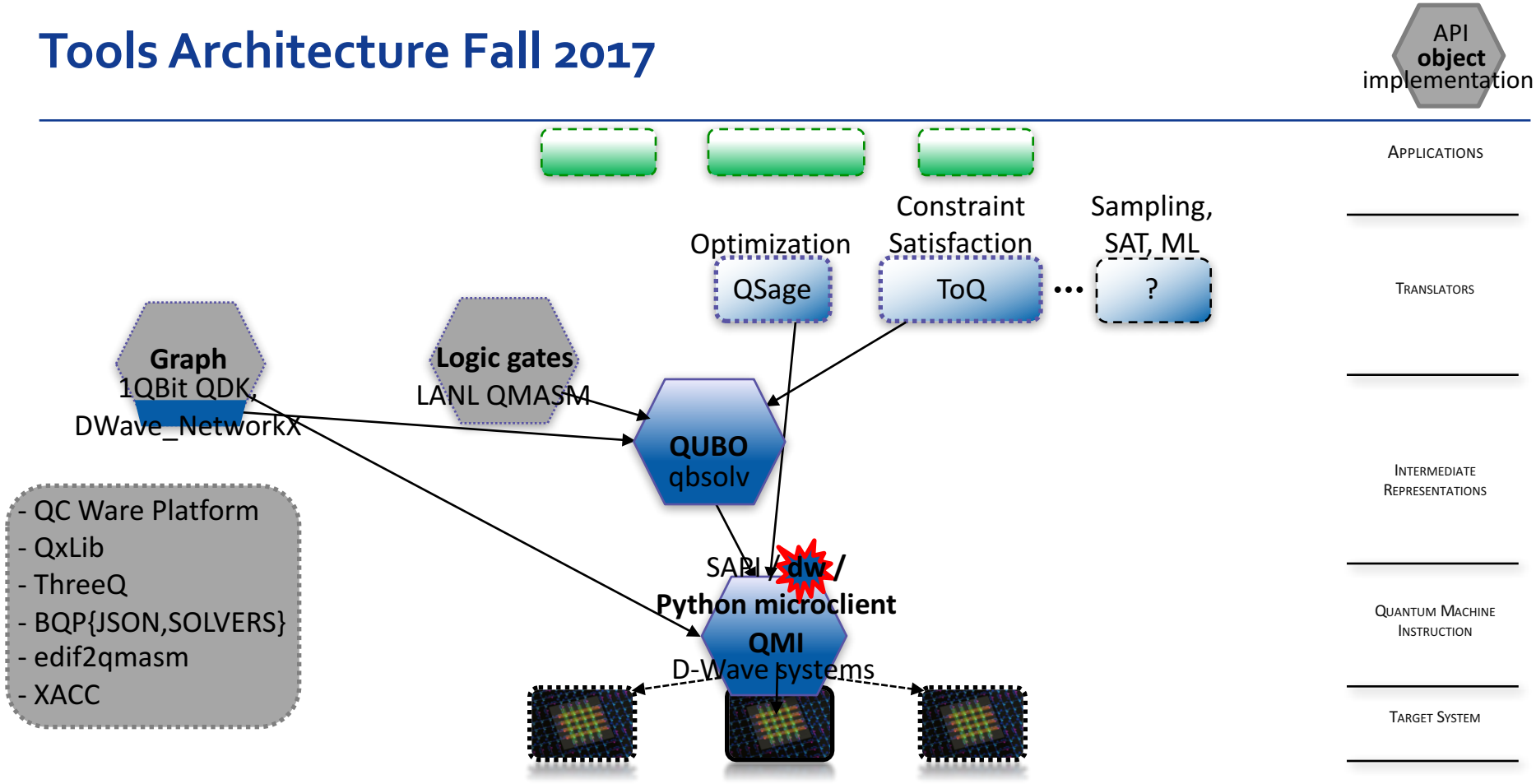
Tools Goals

- Enable applications development
- Enable more smart people to use D-Wave systems
- Foster/collaborate in QA tools ecosystem

Tools Goals

- Enable applications development
  - Enable more smart people to use D-Wave systems
  - Foster/collaborate in QA tools ecosystem

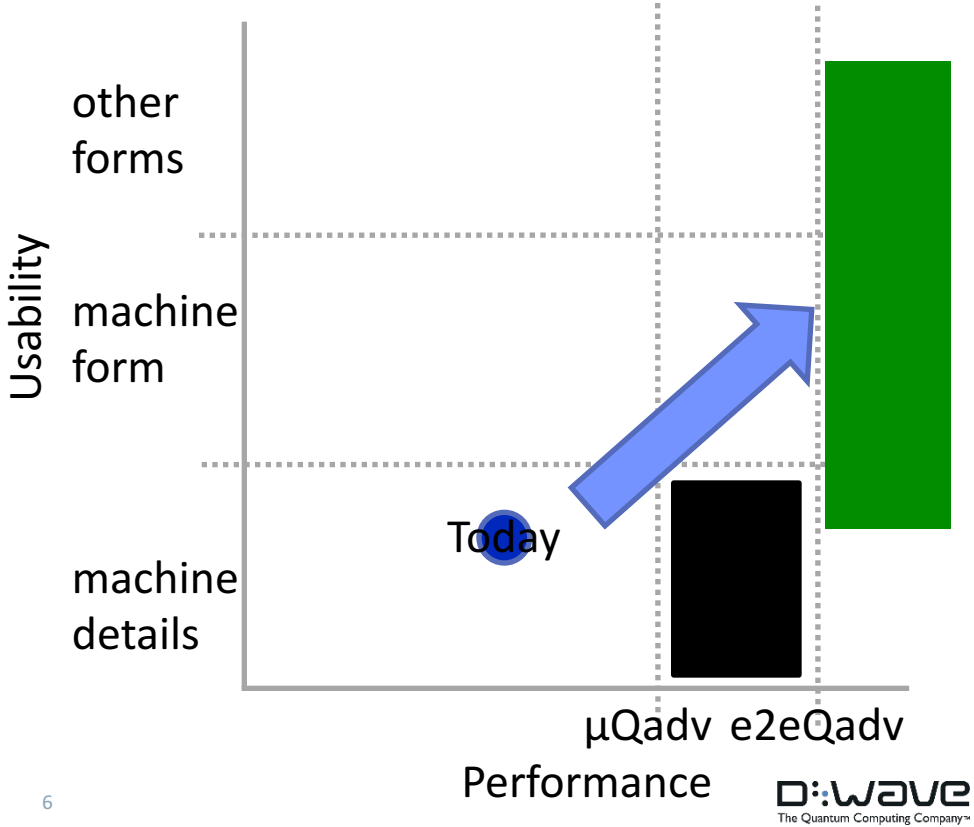
Tools Architecture Fall 2017



D-Wave
The Quantum Computing Company™

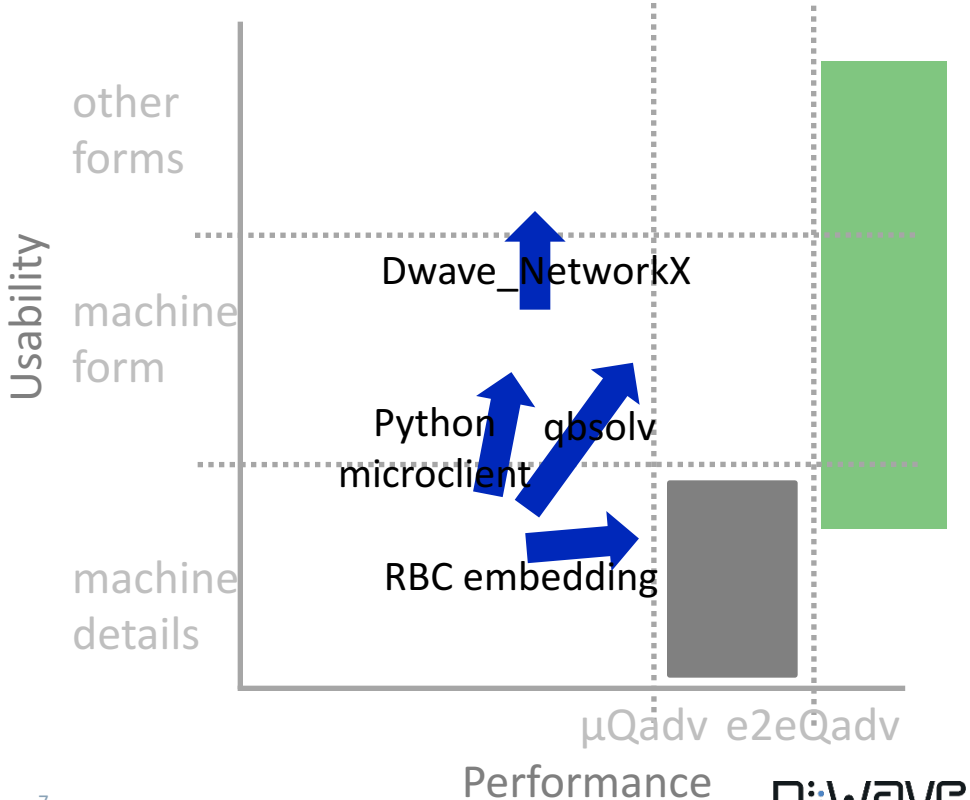
One View of Tools Requirements

- Performance
  - Quantum advantage in the micro (ignoring system overheads)
  - Quantum advantage end-to-end
- Usability
  - Requires app developer to map to machine-specific details
  - Requires app developer to map to system form (e.g., QUBO) but not machine details
  - Does not require app developer to map to system form



Tools Progress Since Qubits 2016

- Performance
  - Quantum advantage in the micro (ignoring system overheads)
  - Quantum advantage end-to-end
- Usability
  - Requires app developer to map to machine-specific details
  - Requires app developer to map to system form (e.g., Ising model, QUBO)
  - Does not require app developer to map to system form



dw

- Command-line tool primarily intended for training
- [New] Build a QUBO problem and solve it via qbsolv
- [New] Build a constraint problem, “compile” it, and solve it via qbsolv
- [New] Library version of core functions, used by qbsolv for faster execution

Python microclient

- Solver API (SAPI) client libraries are closed source
- Having open-source projects depend on closed source packages is difficult
- Python microclient implements a minimal set of SAPI client functionality and is open-source
  - Establish connection, discover solvers, submit QMIs, get return/status
  - Not yet embedding or post-processing
- Recently open-sourced – github.com/dwavesystems/dwave_micro_client

qbsolv

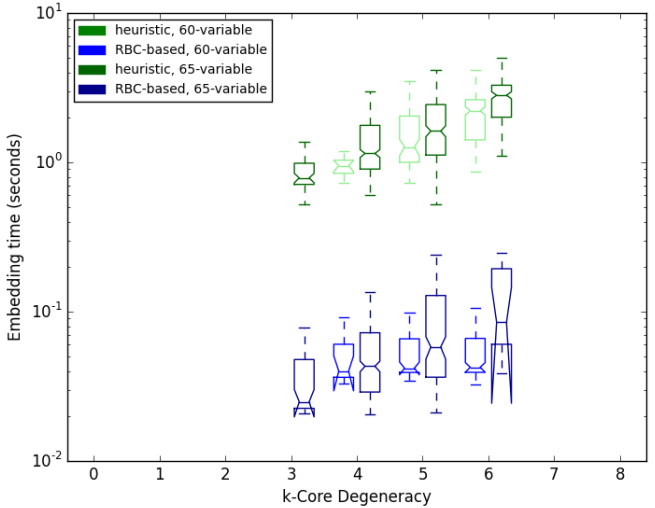
- Hybrid classical/quantum SW solving an arbitrary QUBO by decomposing into chunks suitable for solution on D-Wave system, combining chunk solutions, and iterating
- Performance: classical is state of the art; quantum is slower
- Open-sourced on Jan10; little code contributed so far by non-D-Wave people
  - github.com/dwavesystems/qbsolv for code and technical report
- Active collaboration with Fred Glover, Gary Kochenberger, et al. on better algorithms
- Alternate decomposition algorithm (path relinking) released in qOp2.4 (May)
- D-Wave execution speed improvements and Python interface released in qOp2.5 (Sep)
- Many tools and application efforts use qbsolv
  - Tools: dw, ThreeQ, edif2qmasm/qmasm, ToQ, DWave_NetworkX
  - Apps: graph partitioning and community detection (LANL), traffic flow optimization (VW), terrorist network formation (LANL), refinery scheduling (UCompostella), cluster analysis (BoozAllen), nonnegative/binary matrix factorization (LANL), multiple sequence alignment (<undisclosed>)

DWave_NetworkX

- NetworkX is a graph-analytic package developed at LANL.
- Many common graph algorithms (maximum independent set, max cut, …) are implemented
- DWave_NetworkX re-implements some of those algorithms for D-Wave execution
  - Minimum vertex coloring, min vertex cover, elimination ordering, maximum independent set, maximal matching, max cut, signed social network
- Also adds some useful graph-visualization functions for Chimera-based graphs
- Recently open-sourced – github.com/dwavesystems/dwave_networkx

RBC-based Embedding

- Qbsolv needs a fast, space-efficient embedder for chunks it creates
- RBC == recursive-bisection connectivity, a metric that measures graph density with a notion of distance
- Idea: Generate numerous pre-embeddings; match problem graph with pre-embedding(s) on the fly, using RBC to accelerate failure
- Optimizations: re-order pre-embeddings by prior success, and use k-core-degeneracy of problem graph to predict and so avoid failure to embed
- Embeds 60-65-variable graphs on D-Wave 2X™ (~1.4X more variables than clique previously used) in an average of 20-140ms, depending on problem graph
- Under consideration for possible open-source release

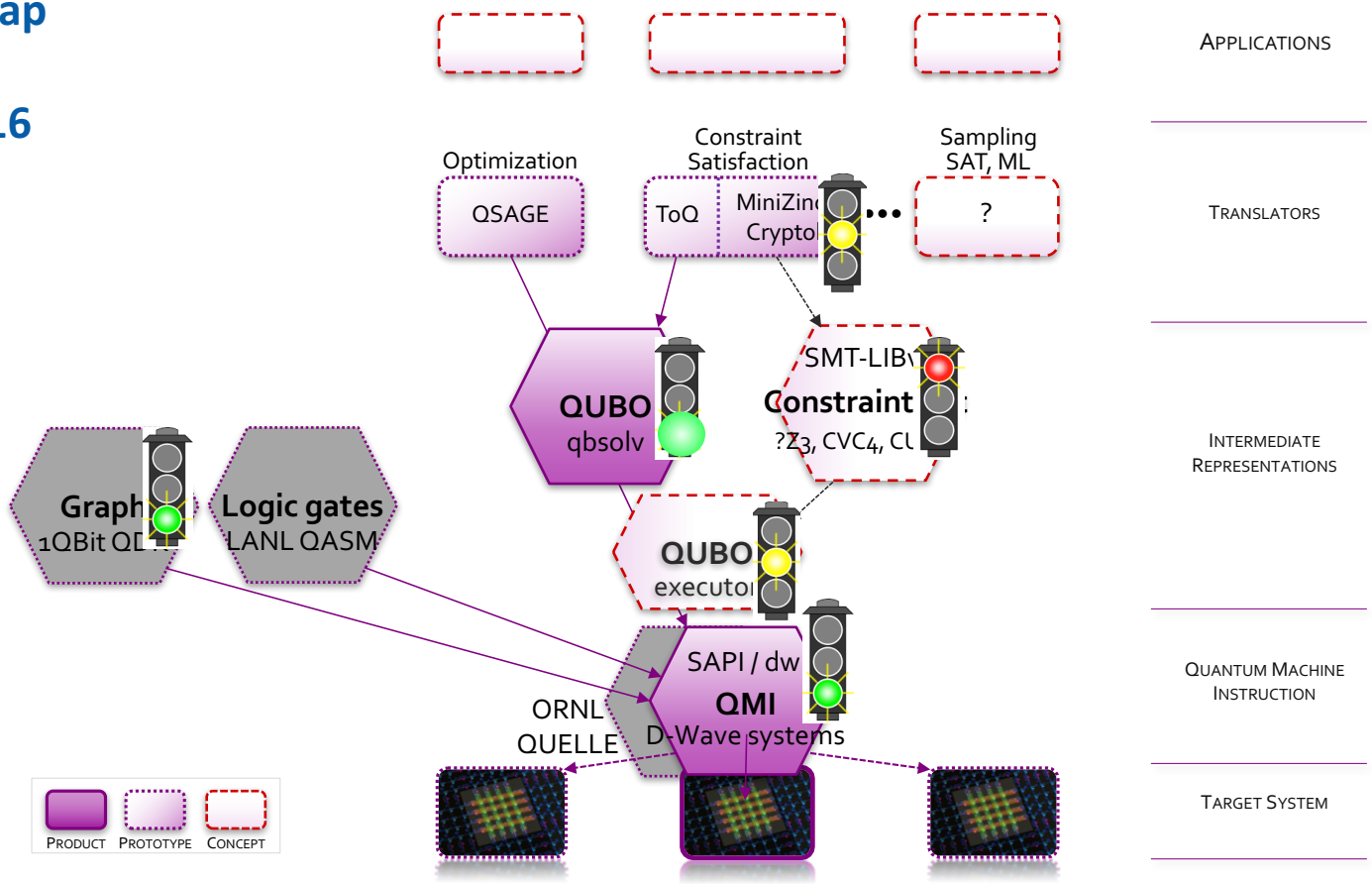


<add slide about emerging tools community?>

D-Wave Software Environment – Fall 2017

API **object** implementation

My roadmap slide from Qubits 2016



Software Requirements for 2022

- Hardware must be radically faster
- Current tools need to deliver hardware performance much more effectively
  - Need much lower QMI latency
  - Need much faster and reasonably efficient embedding
  - Need QMI scheduling that delivers quantum advantage, once attained, to 1 user (then 3, then 10, …)
- Tools need to map arbitrary QUBOs to hardware much more effectively
  - Decomposition and clamping algorithms, faster execution, better use of sampling
- Tools need to connect higher-level interfaces to QUBO execution
  - QUBO is a classical computation
  - App developers use existing languages/interfaces (and solvers) for solving QUBO problems

Need for Much Lower QMI Overhead/Latency

Work in classical units	Total anneal time (ms)	Narrow perf ratio
5	5	1

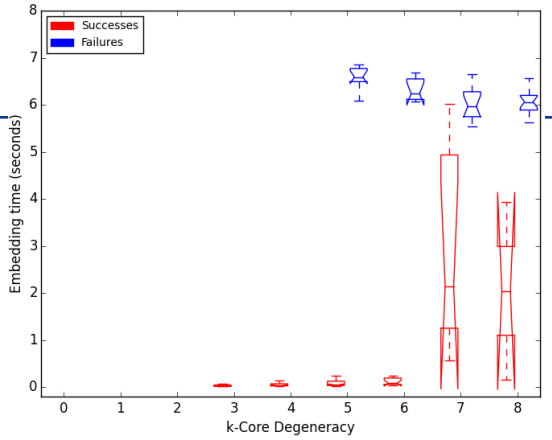
- From a similar argument, embedding (even with more qubits) must execute in O(10ms)

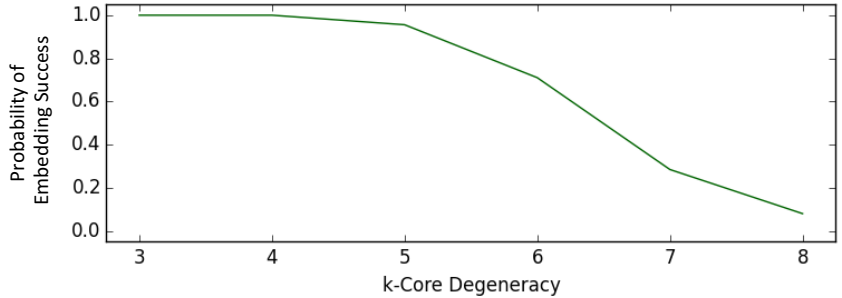
Cause for Optimism: NP-hard Problems Solved in Modern Compilers

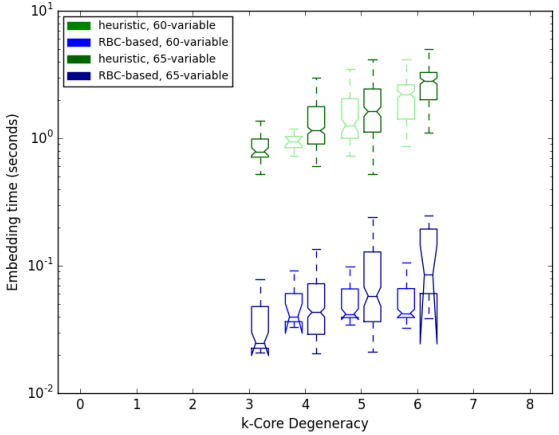
• Graph coloring	Code generation
• Set-weighted covering	Register sufficiency
• Topological sort	Instruction scheduling
• Graph coloring	Register allocation, coalescing, minimizing spill, and reuse
• Minimal vertex covering	Global reference allocation
• Maximum weighted path cover	Array unification
• Multiple graph partitioning	Distributed memory layout

Cause for Optimism: We Learn

- During RBC-embedding development, stuck at ~1.2s for average embedding time
- Time is dominated by failures, which search many pre-embeddings. Want to identify quickly problem graphs that are likely to fail
- Timothy Goodrich suggested k-core degeneracy, which predicts success/failure well
- With k-core-degeneracy, average time shrinks to 20-140ms







Summary

- We have established QUBO/qbsolv as a guidepost for apps and higher-level tool development
- We have established core pieces of an open-source set of tools
- Need to make qbsolv (and other decomposing solvers) run much more efficiently
- Need to connect existing higher-level languages/interfaces to qbsolv and similar
- Absolute times need greater focus
- Lots of bright minds working on this; D-Wave can’t do it alone; collaborations essential