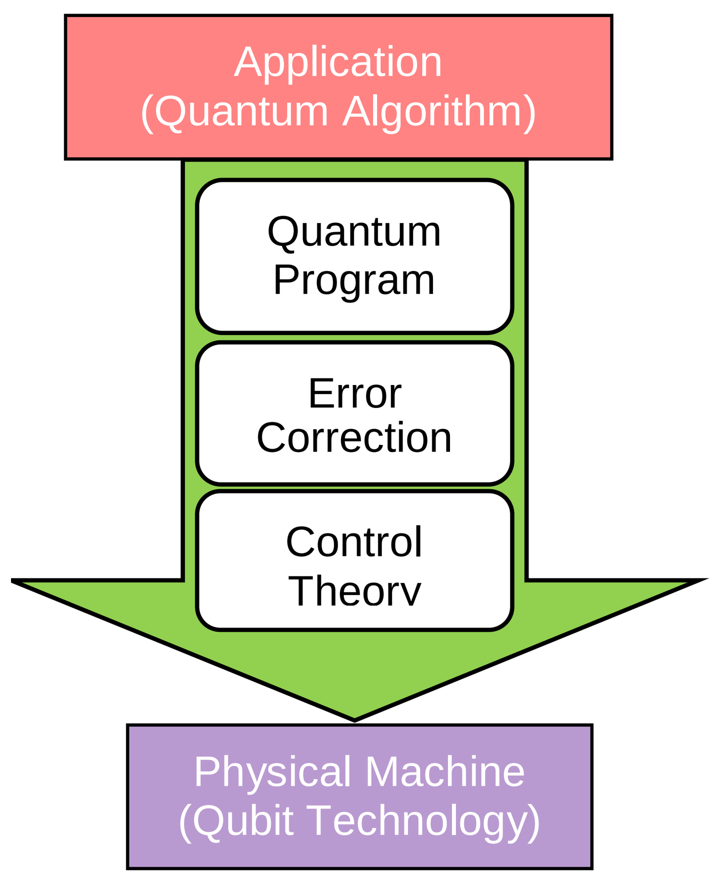
Quantum Computer Science (QCS)

The Gap is Even Larger than We Thought

Program Manager: Dr. Mark Heiligman; E-mail: mark.heiligman@iarpa.gov

A way forward is to estimate the quantum resources needed and time to completion for algorithms of interest.

- Pick a set of quantum benchmark algorithms & protocols for quantum error correction and control
- Leverage classical computer science notions like programming and compilation to render these algorithms as quantum gate operations
- Use assumed characteristics of the qubit types to derive estimates for gates, operations and, eventually, algorithms



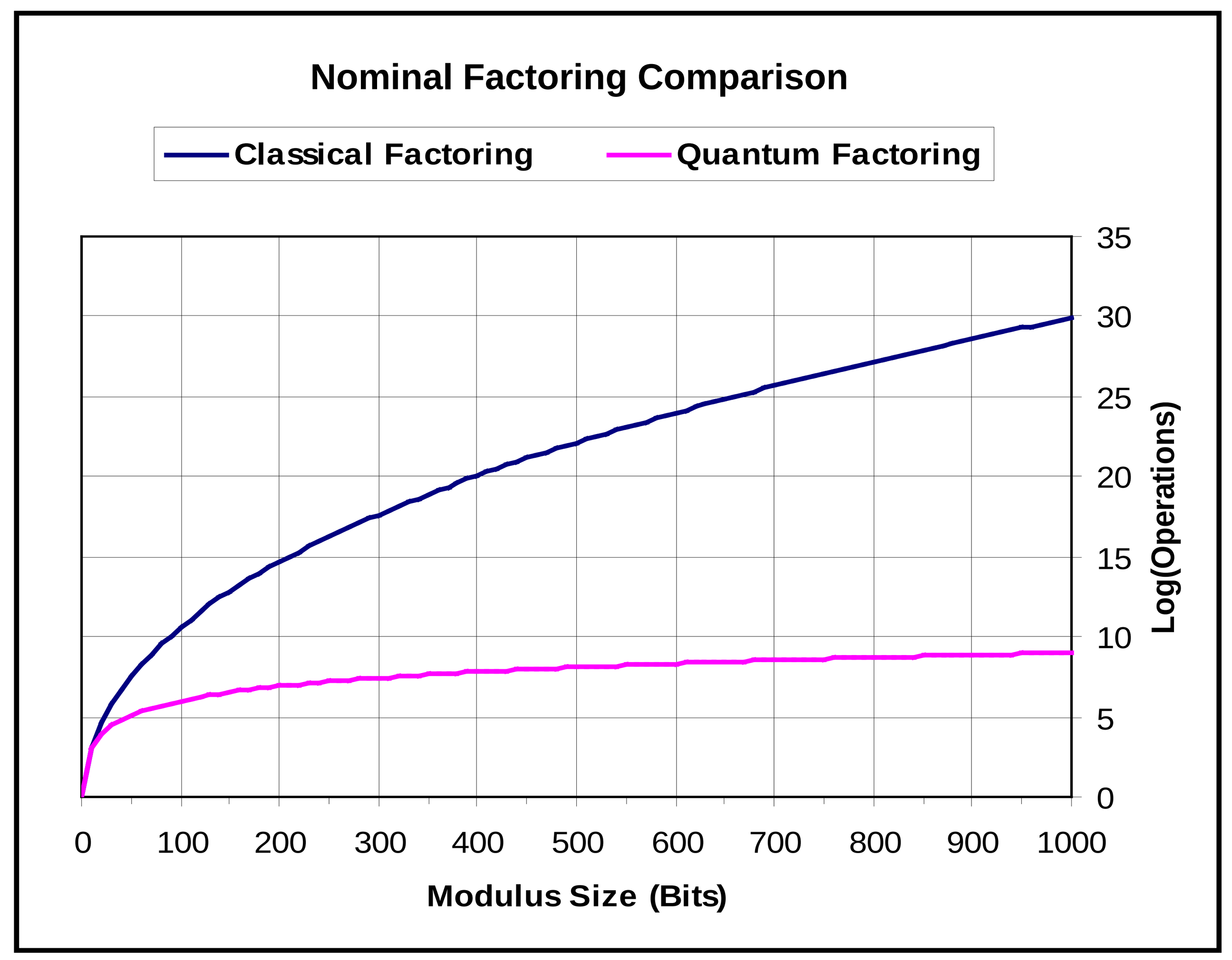
We structured the program into two phases: develop baseline resource estimates; further explore the parameter space.

APPLIED COMMUNICATION SCIENCES · Raytheon BBN Technologies · Georgia Tech · USC University of Southern California

FY10Q3: Benchmark problem set developed
FY11Q4: Program kicked off
FY12Q4: Baseline resource estimates computed; 3 new quantum programming languages developed
FY13Q3: Initial delivery of quantum programming toolbox and new protocols for quantum control and quantum error correction

How can we estimate how big and fast a quantum computer would need to be in order to do anything useful?

- Theoretically quantum computing looks like it could do some important computational tasks very fast
- To be useful a QC needs to do better than a classical computer



Theoretical claims have largely ignored overhead associated with instantiation of algorithms by realistic physical systems.

- QCS looked at annotated algorithms, that is, all of the quantum operations required to implement an algorithm including control, housekeeping, and error correction
- In the absence of such actual systems, QCS posited PMDs that, albeit idealized, will approximate best case performance
- QCS parameterized PMDs so as to be able to look at a range of error correction techniques and algorithms

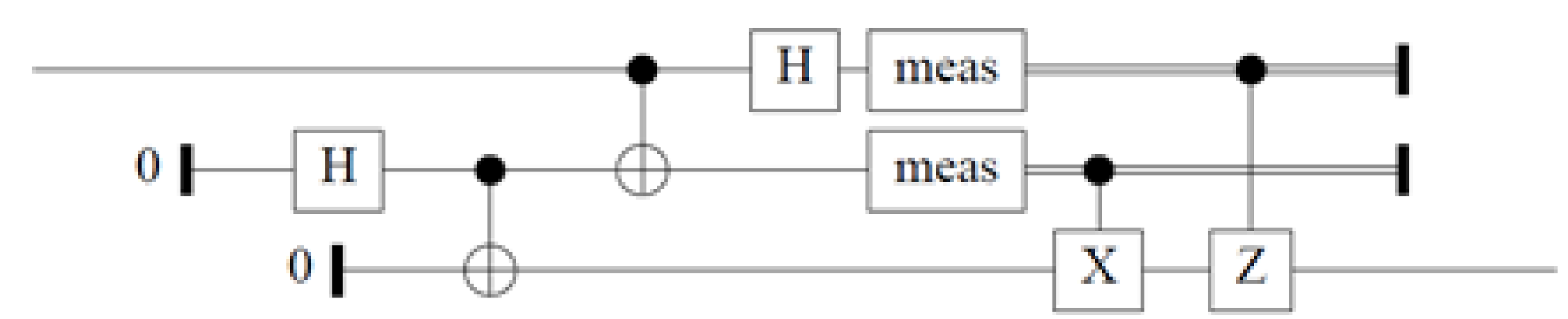
QCS confirmed that, even considering our idealized Physical Machine Descriptions (PMDs), quantum computers will spend most of their time doing error correction.

- Demonstrated that the speed of error correction dictates how fast a quantum algorithm can be executed, **not** the physical gate speed. E.g., Superconducting vs Ion traps.
- Discovered that error correction overhead is overwhelmingly driven by a single gate type.

QCS developed the world's first high level quantum programming language and compilers.

- Developed three quantum computing programming languages and compilers: QUIPPER (ACS), QUAFL (BBN), and SCAFFOLD (USC & GT)

```
import Quipper

plus_minus :: Bool -> Circ Qubit
plus_minus b = do
  q <- qinit b
  q <- hadamard q
  return q

share :: Qubit -> Circ (Qubit, Qubit)
share a = do
  b <- qinit False
  b <- qnot b `controlled` a
  return (a,b)

bell00 :: Circ (Qubit, Qubit)
bell00 = do
  a <- plus_minus False
  (a,b) <- share a
  return (a,b)

alice :: Qubit -> Qubit ->
  Circ (Bit,Bit)
alice q a = do
  a <- qnot a
   `controlled` q
  q <- hadamard q
  (x,y) <- measure (q,a)
  return (x,y)

bob :: Qubit -> (Bit,Bit) -
  > Circ Qubit
bob b (x,y) = do
  b <- gate_X b
   `controlled` y
  b <- gate_Z b
   `controlled` x
  cdiscard (x,y)
  return b

teleport :: Qubit -> Circ
  Qubit
teleport q = do
  (a,b) <- bell00
  (x,y) <- alice q a
  b <- bob b (x,y)
  return b
-- main functions
main_alice =
  print_simple Preview
  alice
main_bob =
  print_simple Preview
  bob
main_teleport =
  print_simple Preview
  teleport
main = main_teleport
```



QCS tools provide a good foundation for further work to understand the expected performance of quantum systems.

- Move beyond the ability to characterize idealized systems by incorporating more complex error models