Why Daml?

Introduction

Daml is a development framework for creating systems of record that securely connect across business processes and data boundaries while retaining privacy, guaranteeing data integrity, and enforcing strict authorization. Allowing such systems to cross boundaries — legal, regulatory, organizational, or other — enables them to form seamless economic networks.

Daml's core idea is to abstract the key concepts of blockchains that allow them to provide consistency guarantees on distributed systems of record. Daml, a virtual shared ledger, governed by smart contracts, takes the place of a concrete blockchain. Adding this layer of abstraction allows Daml to realize a whole range of capabilities and advantages that are sorely lacking in individual blockchain and distributed ledger technologies.

Most importantly, it decouples applications from underlying storage and consensus, making Daml applications portable and enabling unprecedented privacy and interoperability without trading off consistency. The way Daml abstracts away the underlying blockchain, distributed ledger, or database to expose a virtual shared ledger is analogous to how high-level programming languages abstracted away microprocessor architectures in favor of memory and logic abstractions in the 60s and 70s, or how SQL and ODBC/JDBC abstracted away relational database systems to popularize the CRUD model in the 90s. In the same way, these past developments revolutionized how we built applications for mainframes and the web. Daml aims to revolutionize how we build applications across entities or other boundaries.

One of the reasons abstraction layers drive innovation is that they move the focus from the deployment target to the application that runs on top. Developers are exposed to less complexity and can focus on solving business rather than technical problems. Daml fully embraces this. Its purpose-built smart contract language provides the primitives to safely and easily express complex business logic with its associated privacy and authorization concerns. On top of that, an entire application stack allows developers to integrate Daml applications into holistic enterprise solutions and existing systems.

Daml's core idea is to abstract the key concepts of blockchains that allow them to provide consistency guarantees on distributed systems of record.

The combination of portability, unmatched capabilities, and ease of development simultaneously future-proofs investment, opens up new use cases, and de-risks projects. In what follows we will describe in more detail some key issues facing enterprise blockchain projects, how Daml delivers the benefits described above, and how it compares to other solutions in key areas like application development, deployment, privacy, and interoperability.

Distributed Ledgers for Enterprise

Blockchains solve one of the fundamental problems facing every distributed application: How do you provide every node in the network a consistent shared state?

The importance of this is illustrated by the double-spend problem in the use case of digital money. Alice has $100, and both Bob and Carol sell bikes for $100. Alice may try, maliciously or by error, to spend her $100 twice to buy both bikes. Consistency means that the entire network comes to the same conclusion on who owns the $100 when Alice does so.

Before blockchains, this was solved almost universally by a combination of three mechanisms:

1. Centralized "master" record of the state of the system. E.g., the land registry of a country, or bank accounts.
2. Processes to consistently update two or more such "master" records at the same time. E.g., a transfer from an account at one bank to an account at another bank.
3. Reconciliation processes to detect and correct inconsistencies due to implementation errors of the prior two mechanisms.

The complexity of the processes in point 2 and the associated reconciliation in point 3 make such systems slow and expensive to run, which translates directly into significant costs for businesses. The guaranteed system consistency offered by distributed ledgers offers significant savings and speed-up of business processes.

In their original design, blockchains are public and permissionless, meaning anyone can connect to them, everyone can see all data, and all participants are equal. None of these properties are suitable for the majority of enterprise applications, so systems geared towards enterprise and offering different degrees of privacy and permissioning have emerged. They employ a spectrum of different approaches to application development, privacy, and interoperability. However, all current major blockchain and distributed ledger technology (DLT) platforms fall short in several areas:

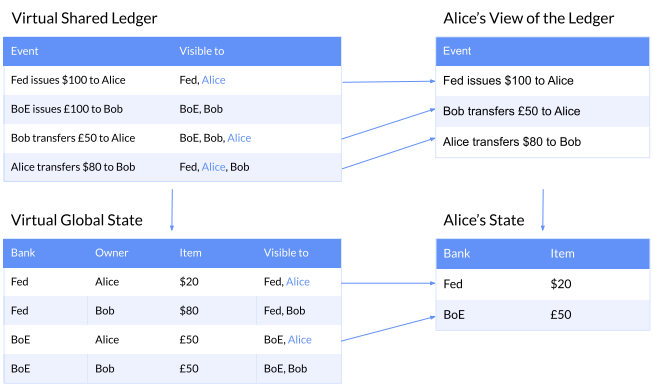
- Programming these systems is difficult and error-prone.
- There is insufficient support for integrating blockchain applications into full solutions.
- Even to start developing an application, one needs to commit to a single deployment target, resulting in applications that are then tied to that target.
- Heavy trade-offs between privacy, security, and consistency are required. These trade-offs are often implicit and are not clear to the developer who needs to make them.
- Interoperability relies on standardizing on a single blockchain or distributed ledger technology or requires careful planning and application development.

Daml solves these as well as other problems within the enterprise blockchain space. Let's now take a deeper look at Daml.

An Overview of Daml

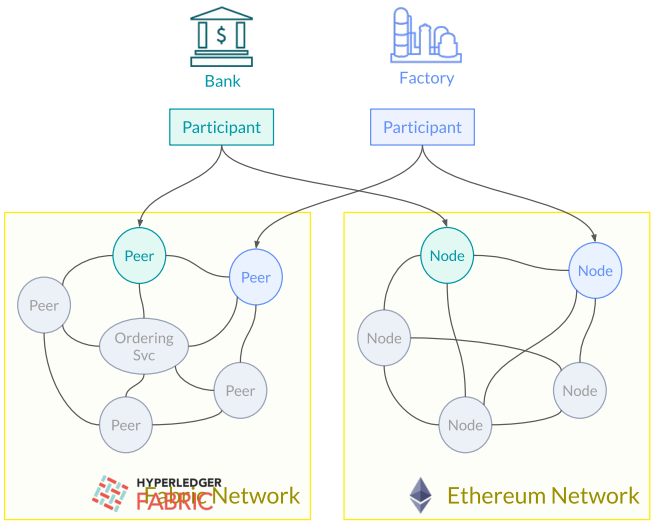
The open source Daml smart contract language, described in more detail below under "Daml Smart Contracts," forms the core of the Daml application stack. The language is a purpose-built Domain Specific Language (DSL) designed to encode the shared business logic of the application. It provides developers primitives to safely and succinctly describe core concerns like data types, privacy, and authorization rules without concerning themselves with the ultimate deployment target. In terms of the virtual shared ledger, it expresses who may write what events, and to whom they are distributed.

Daml abstracts the event sourcing model employed by blockchains and distributed ledgers. Event sourcing means that the current state of the system can be computed from the log of past events. Instead of keeping a global event log, Daml provides each stakeholder a consistent view on a virtual shared event log, called the virtual shared ledger, while distributing data on a strict need-to-know basis under the hood. This creates a system in which each participant has a consistent



view of the (virtual) global system state, consisting of exactly the data they are entitled to. The virtual shared ledger is described in Daml's Ledger Model and implemented in the Daml smart contract language and associated Ledger API.

The Ledger API is the primary API to interact with Daml smart contracts and the virtual shared ledger. It is a high-performance reactive gRPC-based streaming API. It allows applications to subscribe to events that they are permissioned to view, get their current state, or submit commands to write new events. As such, the Ledger API and smart contracts are the primary mechanisms for abstracting away concrete blockchains or databases to instead develop against a virtual shared ledger. Any node in a network that exposes Daml's Ledger API—and thus gives access to the shared ledgers—is called a participant node as it allows participation in the network.



To enable an infrastructure like a blockchain or database to run Daml applications, it needs a Daml Driver that allows a matching participant node to connect into the network. Daml Drivers do not store any data and are typically deployed per node of the underlying blockchain or database. There are already numerous drivers, both open source and commercial, available in the Daml ecosystem, with more under development. See daml.com for an overview.

The deployed Daml smart contracts as well as everything on top of the Ledger API are considered part of a Daml application. Daml is a suite of tools that assists in the development of full-stack Daml applications. Daml contains an SDK that includes an Integrated Development Environment (IDE) for smart contract development, a suite of runtime components, and integration libraries spanning all the way to user interfaces.

With this picture in mind, we can take a look at how Daml is able to provide substantial benefits over bare-bones distributed ledgers.

Programing Enterprise Blockchains

All blockchain systems are based on an immutable and verifiable log of transactions. The state of an application is computed from that transaction log. Where blockchain systems differ profoundly from each other is in what transactions and resulting state look like, as well as in how client applications interact with the ledger. The three most prevalent approaches are the following:

Consensus-only solutions like Hyperledger Fabric, Hyperledger Sawtooth, and Tendermint have low-level APIs that allow external processes or modules to submit and validate transactions. Other than metadata needed for the consensus algorithm, transactions are opaque blobs. That means both the shape of transactions and the resulting state are left to the developer.

In Unspent Transaction Output (UTXO) solutions like R3's Corda, state and transactions have a prescribed format, which user

Building applications directly against the low-level API of consensus-only solutions or using the inbuilt transaction and validation mechanisms of a UTXO solution ties an application irrevocably to the underlying blockchain.

code has to construct and validate. Validation is done using an inbuilt virtual machine.

Ethereum derivatives like Hyperledger Besu and Quorum have a state consisting of global, replicated state machines with transactions consisting of method calls on those state machines.

Building applications directly against the low-level API of consensus-only solutions or using the inbuilt transaction and validation mechanisms of a UTXO solution ties an application irrevocably to the underlying blockchain. It also means that languages and tools designed for traditional application development have to be adapted to fit the requirements and programming paradigms of distributed ledgers. This requires great skill and care by developers and a deep understanding of the underlying blockchain or ledger technology.

A further problem with the consensus-only or UTXO models is that transaction validation has to be deterministic to avoid inconsistencies (forks). Accomplishing this is possible in one of three ways: Use a custom language and VM for validation, like Bitcoin Script, compile a common language like C++ to a deterministic bytecode like WASM, or modify an existing VM like the JVM to be deterministic. All three approaches have the problem that transaction construction and validation use different code-paths, putting the burden on the developer to ensure that only correct transactions get constructed. This is made more difficult if the two use different languages.

For that reason, smart contract development in the style of Ethereum has proven itself to be the winning model for application development on blockchains, but Solidity, Ethereum's de facto smart contract language, is deeply tied to the mechanics of the Ethereum blockchain. Furthermore, the model of transacting via method calls to globally replicated state machines has known issues with security, privacy, and scalability.

Daml Smart Contracts

Daml contracts take the best parts of the UTXO model, Ethereum-style smart contract model, as well as typesafe and pure general-purpose languages and combine them into a unique blend of usability and safety suitable for the task of creating critical and complex applications.

The interaction with Daml contracts is similar to interaction with Ethereum-style smart contracts in that each active contract on the ledger gives rise to a number of choices that look and feel a bit like methods on an object. As a result, transaction construction and validation use the same code, and Daml inherits the familiarity and scalability of the smart contract model.

Rather than being globally replicated state machines, each contract is entirely immutable, meaning these choices can only archive existing contracts or create new ones, giving the safety and simplicity of UTXO models. Security issues like reentrancy bugs or unexpected results due to race conditions are eliminated. Daml ledgers inherit the simple-to-understand notion of state from UTXO ledgers: The current state of the application is simply the set of currently active contracts.

In addition, Daml is unique in that it exchanges ledger native concepts like addresses, cryptographic signatures, transactions, and similar with high-level concepts that matter to applications: Data ownership, visibility, permissions, write authorization, and the like. This allows for seamless extensibility and upgradability of contracts without any special planning or processes beyond usual application development.

Building Client Applications

Smart contracts running on a blockchain or database are only the very core of a full smart contract-based solution, and the majority of work in any project tends to be the integration. Thus, suitable APIs, integration components, and libraries are key to any successful project.

None of the major enterprise blockchains have a coherent application stack beyond exposing low-level functionality in a small number of languages or using a proprietary API. As a result, the application developer is forced into using the language environment chosen by the developers of the blockchain and must build their application in that environment from low-level APIs and a mix of inconsistent tools and libraries. For example:

- Fabric chain code is supported in JavaScript and Java only, and the developer has to program directly against the low-level chain code API.
- In Tendermint, the developer can choose between using the even more low-level ABCI interface to build an application using the language of their choice or going with Cosmos to get something slightly higher-level but then be locked into using Go.
- In Corda, everything, including the RPC client, is tied to the Java (or rather JVM) ecosystem and boils down to exposing ledger functionality in Java rather than integration with wider solutions.

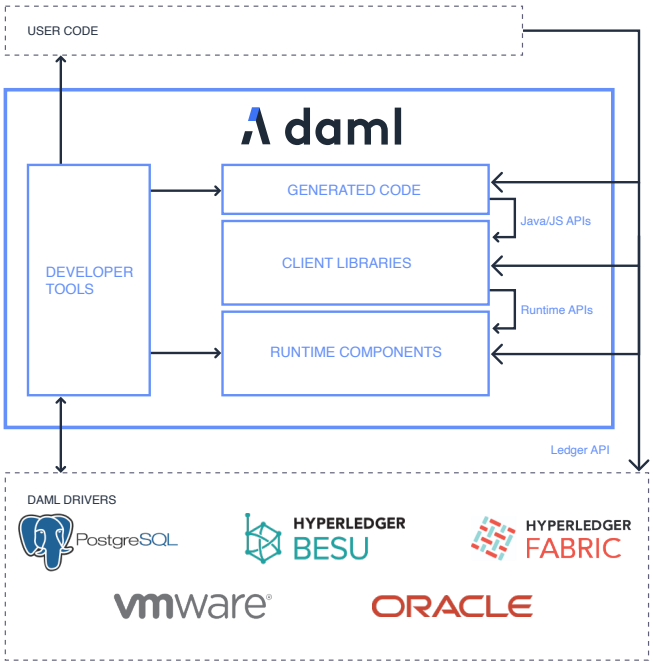
Ethereum has by far the richest ecosystem of tooling, but it is piecemeal and primarily geared towards development against the public main net.

Daml Tooling

Daml follows the philosophy that a smart-contract based solution should embed smoothly into any enterprise context rather than force the developer to adapt their tech stack to the chosen blockchain or database. However, the solution should also support the developer across the entire application stack to enable rapid prototyping and go-to-market. Supporting an entire application stack in every language and framework is not feasible, which is why Daml follows a layered approach.

Daml provides a single coherent stack of APIs, components, and libraries that covers the entire integration space all the way to and including UIs. Within that stack are several stable and broadly supported APIs that allow developers to branch off into their own languages or frameworks with ease. The different components and layers of Daml communicate only through these public APIs, which means any functionality provided by a Daml component is also supported by the public APIs of the next-lower-level layer of the stack.

To make this flexible and modular architecture work in as many contexts as possible, Digital Asset chose the most broadly used and accessible technologies, standards, or frameworks fit for their respective purpose: gRPC for high performance streaming, HTTP/JSON/WebSockets for high-level APIs, Java and



Testability And Deployability

To test or deploy an application built natively for a blockchain, you need to spin up an entire enterprise-grade network. That tends to be an expensive and complex undertaking affecting both the development process as well as the cost of deploying pilots. Frameworks like Hyperledger's Blockchain Automation Framework or blockchain-as-a-service platforms can help with this to a limited extent, but do not come close to the development and deployment experiences of general-purpose application frameworks or platforms.

Tools like Truffle Ganache, a small in-memory implementation of Ethereum, smooth the development process somewhat as they allow developers to spin up an ephemeral deployment target quickly on their local machine or in Continuous Integration (CI). But Ganache simulates the Ethereum main net and is thus not entirely suitable for testing enterprise applications.

Because Daml is not tied to any particular deployment target, it can take both local development ledgers and easy service-based deployment targets to the next level. Daml is able to evaluate smart contract code in real time in the IDE, giving immediate feedback to developers and enabling fast test-driven development practices. It is able to provide not only a local test ledger but to spin up an entire application platform including all Daml components on the developer's machine.

When it comes to deploying a production system, the deployment target can be chosen to suit the application, with options ranging from managed PaaS solutions like project:DABL, via blockchain- or database-as-a-service offerings to on-premise solutions. Many Daml Drivers can even be deployed to already running blockchains or databases.

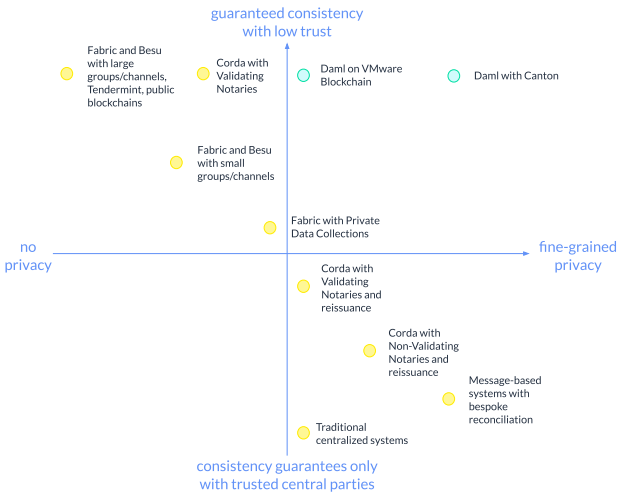
Once a target deployment environment is chosen, tests that were run by the developer and CI can be rerun against a production-like test or staging environment.

Privacy

All of today's blockchains require users to make heavy trade-offs between consistency and privacy. Simply put, each deployment can choose strong privacy or strong consistency, but not both at the same time.

Fabric, Besu, and Quorum allow transactions to be made between restricted sets of parties via private transaction managers, but those sets can never change. An agreement private to Alice and Bob cannot be used in a transaction involving Alice, Bob, and Carol. The only exception to that is that private transactions may read public information.

- To transact safely amongst a group using the inbuilt consistency mechanisms, all involved parties see all involved data.
- To get better privacy, data needs to be transferred from one set of parties to another using off-ledger processes. Developers need to ensure consistency themselves.



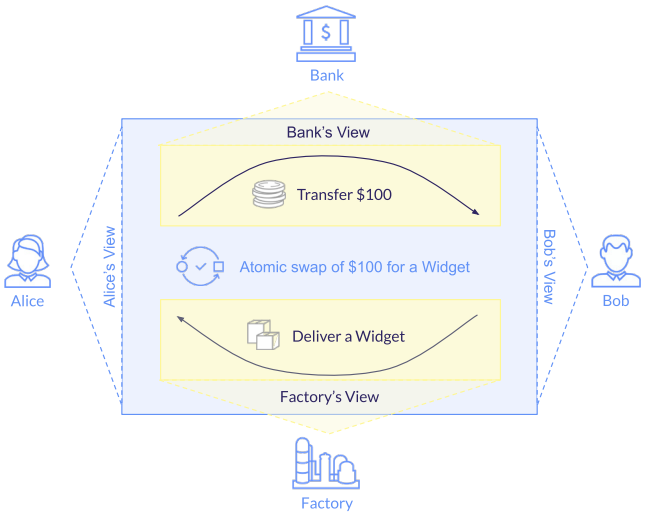
In Corda, transactions are only seen by those parties that are involved in that transaction. However, every transaction includes the full history of its input contracts. So if Alice issues a token to Bob, Bob transfers it to Carol, and Carol transfers it to Duncan, Duncan sees the entire chain Alice -> Bob -> Carol -> Duncan. The common workaround — reissuance after every transfer — compromises consistency. Furthermore, networks need to make a choice between validating and non-validating notaries. Validating notaries see everything, thus achieving consistency by forgoing privacy. Non-validating nodes preserve privacy, but are open to so-called Denial of State attacks, which can be used to maliciously lock up applications.

Subtransaction Privacy In Daml

Daml's privacy model is considerably more granular than either the "channel" model of Fabric and Enterprise Ethereum or Corda's transaction privacy. Daml transactions are composed of subtransactions and every subtransaction has its own set of witnesses that are entitled to see that part of the overall picture. For example, if Alice and Bob want to swap $100 at a Bank for a Widget at a Factory, they can do so atomically with the Bank only seeing the cash transfer and the Factory only seeing the transfer of the Widget. The set of witnesses depends only on the current transaction, not on ledger history, making this model predictable and safe to work with. This privacy is provided by data minimization, sending only those parts of a transaction to a participant node that it is entitled to see.

This strong privacy comes without any trade-off on consistency. The witnesses of Daml transactions are computed to ensure that all relevant parties can validate enough of every transaction to guarantee ledger integrity.

The full Daml privacy model is provided in all integrations from the point of view of consumers of the Ledger API. The degree to which privacy can be upheld against operators of infrastructure nodes or Daml Drivers depends on the capabilities of the underlying ledger. When an underlying system cannot support Daml's full privacy model, Daml makes this choice explicit, enabling security teams to make an apples-to-apples comparison of the security attributes of the different blockchains. In many cases, using a combination of data minimization and cryptography, Daml can achieve better privacy than the deployment target can offer natively, without impacting integrity. Naturally, if Daml is deployed to a centralized SQL database, the database operator will have access to at least encrypted versions of the transactions. But thanks to the portability of Daml applications and the wealth of integrations available, suitable Daml Drivers exist to uphold full privacy in almost all trust contexts.



Interoperability

The blockchain vision is one of a seamlessly interconnected network of business across the globe, a network in which business processes can be composed of building blocks built by others. For the technology to fully deliver on this vision and provide long-term value from the solutions that are being built today, the assets and processes digitized must be able to form the building blocks of future solutions. Future-proofing investment in this way and enabling the network effects promised by blockchain technology require seamless application interoperability.

Smart contracts usually compose nicely but only on a single blockchain. Thus one approach to interoperability is technological alignment. If you consider blockchain analogous to HTTP on TCP/IP and everything that runs on top as the application layer, it is possible to imagine that the world may align around one enterprise blockchain. If, however, you think of a blockchain of anything higher level, like a database or application server, that becomes unlikely. There are few high-level standards that have truly universal adoption.

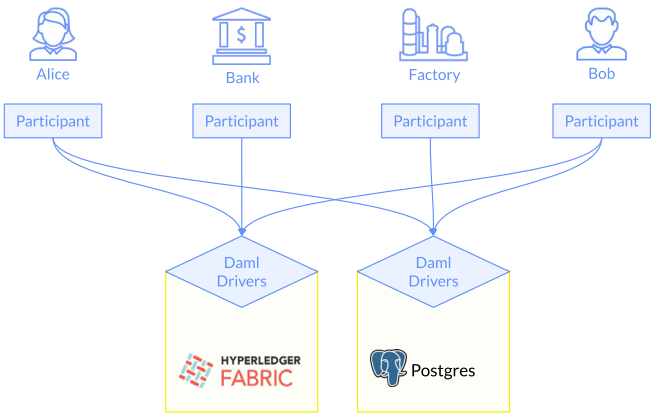
Thus the only realistic way to enable application interoperability is to enable interoperability betweenblockchains and between blockchains and other systems. There are currently three well-understood approaches to this:

1. Oracle systems, which also add interoperability with external systems. These are effectively trusted intermediaries that make external data available on a blockchain.
2. Mutual on-chain validation solutions like Cosmos' Inter Blockchain Communication protocol rely on implementing clients for blockchains on other blockchains. That is, there would be a smart contract on chain A that can validate that a certain action has taken place on chain B, and vice versa. This allows for data transfer between chains, but developers must plan for interoperability when building the application and must write code that specifically handles the mechanics of interoperability throughout.
3. Baselining or Pegging, where a central, often public, blockchain is used to store only references or cryptographic proofs. The Baseline Protocol and Cosmos are projects that use this model, using public chains as the center of a hub-and-spoke model, respectively.

None of these three approaches offer real smart contract interoperability across chains. A contract residing on chain A cannot call a "method" on a contract residing on chain B. Building applications that span chains takes a lot of skill, effort, and overhead.

The Canton Protocol

All three approaches above are easily implemented in Daml. In addition, second generation Daml Drivers use the Canton protocol to reduce reliance on the underlying chain. This can be thought of as running Daml as a Layer 2 protocol or doing something akin to baselining but without the application-specific architecture concerns. Not only are Canton-enabled Daml Drivers able to provide full sub-transaction privacy without trading off consistency, but they can also offer true smart contract interoperability across networks.



With the Canton protocol, a corresponding participant node is able to connect to multiple Daml networks at the same time, while still exposing a single Ledger API to the outside world. Transactions are now able to span one or more Daml networks, with contracts created in one network able to call choices (Daml's equivalent of methods) on contracts created in a different network. From the Daml contract and client application perspective, there is a single virtual ledger spanning all connected networks. No special planning is required for interoperability. A Daml application can be deployed not only on any Daml-network thanks to portability, but actually across multiple Canton-enabled networks at the same time, providing unprecedented interoperability.

In short, Alice and Bob can trade cash for widgets without changing their application even if the Bank and Factory do not operate on the same networks.

Summary

To recap, what is Daml?

- Daml is an open source technology that allows the development and deployment of applications that cross boundaries — organizational, legal, regulatory, etc.
- Daml does so using a purpose-built smart contract language, full-stack application and development tooling, and drivers for blockchains, distributed ledgers, and databases.
- Daml solves many of the critical problems facing enterprise blockchain. It:
- Provides a high-level programming model for distributed ledgers with the right abstractions for business applications
- Provides support across the entire solution and software development lifecycle, not just for the low-level blockchain/smart contract slice
- Is platform-independent, removing lock-in and allowing for cheap and fast go-to-market
- Provides best-in-class privacy without compromising integrity
- Offers true smart contract interoperability between different networks and deployment targets

If you want to learn more:

- Learn Daml using interactive tutorials.
- Install Daml and start developing.
- Talk to us on the Daml forums.
- Contact Digital Asset sales.