Doma Quad Auction Protocol

DQAP 1.0

DI Networks

Sep. 2025

Doma Quad Auction Protocol

A comprehensive auction ecosystem for domain NFTs featuring four specialized systems: Hybrid Batch Auctions for portfolios, Premium Single Domain Auctions with betting, Auction-Backed Lending for liquidity, and Domain Voting Contests for community engagement.

Innovation

The Quad Auction Protocol represents a paradigm shift in NFT auction design by integrating **four complementary economic mechanisms** into a single, cohesive ecosystem. Unlike traditional single-mechanism auction platforms, our protocol creates **network effects** where each system enhances the others:

🏗️ Multi-Layer Economic Architecture

- **Auction Layer**: Traditional price discovery through Dutch auctions
- **Credit Layer**: Auction-Backed Lending introduces borrowing/lending mechanics
- **Gamification Layer**: Loyalty rewards and voting contests drive engagement
- **Social Layer**: Community-driven discovery and validation

🔄 Interconnected Value Flows

- **Liquidity Synergy**: Lending provides upfront capital, enabling more auctions
- **Engagement Flywheel**: Voting contests boost visibility, increasing auction participation
- **Reward Multiplier**: Successful auctions fund loyalty programs and betting pools
- **Risk Mitigation**: Collateralized lending reduces counterparty risk for all participants

🎯 Market Efficiency Innovations

- **Fractional Access**: Small investors participate in premium portfolios through batch auctions

- **Dynamic Incentives**: Reverse royalties and gamified rewards optimize participation
- **Price Discovery**: 4-tier betting provides market sentiment and validation
- **Time Optimization**: First-bid-wins and automated settlement reduce friction

🌐 Ecosystem-Wide Benefits

- **10x Participation**: Multiple entry points for different user types and capital sizes
- **Self-Reinforcing Growth**: Each system creates demand for the others
- **Capital Efficiency**: Lending unlocks liquidity, betting creates yield opportunities
- **Community Governance**: Voting contests align protocol incentives with user preferences

⚡ Technical Innovation

- **Modular Architecture**: Independent systems that can operate separately or together
- **Cross-System Integration**: Smart contracts communicate across different mechanisms
- **Automated Settlement**: Trustless execution of complex multi-party agreements
- **Scalable Design**: Each system can grow independently while benefiting from network effects

Result

A complete domain trading ecosystem that transforms one-dimensional auctions into a multi-dimensional marketplace where trading, lending, betting, and community engagement create unprecedented liquidity and engagement.

Four Specialized Auction Systems

System 1: Hybrid Batch Auctions

Problem

Traditional auctions handle domains individually, making it hard for large holders to liquidate portfolios efficiently and excluding small buyers from participating in premium bundles.

Solution

Hybrid Batch Auctions combine multiple domains into single Dutch auction curves with fractional ownership support, gamified bidding mechanics, and dynamic pricing incentives.

Key Features

- **Batch Portfolio Trading**: Auction multiple domains as a single bundle with shared pricing curve
- **Fractional Ownership**: Buyers can bid for specific percentages of the portfolio (e.g., 10% stake)
- **Gamified Bidding**: Soft bids with auto-conversion thresholds and loyalty rewards
- **Reverse Royalty Engine**: Dynamic royalties starting at 0% and increasing over time
- **Anti-Spam Bonds**: Refundable deposits (0.2%) prevent malicious bidding

Benefits

- **Liquidity for Sellers**: Large holders can liquidate entire portfolios efficiently
- **Accessibility for Buyers**: Small investors can participate in premium bundles fractionally
- **Higher Participation**: Gamification encourages early engagement and prevents last-minute sniping
- **Dynamic Incentives**: Reverse royalties create urgency without fixed high fees
- **Community Building**: Loyalty rewards create sticky, engaged user base

Batch Dutch Auctions

- Auction multiple domain NFTs as a portfolio

- Fractional ownership support (buy specific token counts)
- Linear price decay over time
- Reserve price protection

Gamified Bidding System

- **Soft Bids**: Intent-based bidding with auto-conversion
- **Hard Bids**: Immediate purchase at current price
- **Bonds**: 0.2% refundable deposit prevents spam
- **Loyalty Rewards**: Time-weighted points for early engagement
- **Sale-Gated**: Rewards only distributed on successful auctions

Reverse Royalty Engine

- Dynamic royalties starting at 0%
- Increases per block to incentivize quick trades
- Optional feature for secondary sales
- Automatic distribution to original creators

System 2: Premium Domain Auctions with Betting

Problem

Batch auctions serve portfolios well, but premium single domains need focused attention and additional engagement mechanisms. Traditional auctions lack sophisticated betting systems that could create yield opportunities and price discovery.

Solution

Independent single-domain Dutch auctions with integrated 4-tier price betting system, using commit-reveal protocol to ensure fair wagering and prevent market manipulation.

Key Features

- **Independent Single-Domain Auctions**: Dedicated Dutch auctions for premium domains
- **First-Bid-Wins Mechanism**: Immediate auction completion on first bid for efficiency
- **4-Tier Price Betting**: Parallel wagering on auction outcomes (Above High, High~Low, Below Low, Uncleared)
- **Commit-Reveal Protocol**: Hidden bets prevent market manipulation and sniping
- **Configurable Price Thresholds**: Seller-defined high/low boundaries for betting categories

Betting Categories

- **Category 3**: Final price > High Price (premium outcome)
- **Category 2**: Low Price ≤ Final price ≤ High Price (expected range)
- **Category 1**: Final price < Low Price (bargain outcome)
- **Category 0**: Auction fails to clear (no sale)

Revenue Distribution

- **90%**: Winning bettors (pro-rata by stake)
- **5%**: Seller (liquidity premium)
- **3%**: Winning buyer (price discovery bonus)

- **2%**: Protocol treasury

Benefits

- **Independence**: Separate system for different use cases from batch auctions
- **Fairness**: Hidden commitments prevent market manipulation and sniping
- **Incentives**: All participants (sellers, buyers, bettors) receive rewards
- **Engagement**: Creates yield opportunities around premium domain auctions
- **Efficiency**: First-bid-wins mechanism reduces auction duration

System 3: Auction-Backed Lending for Domain Auctions

Problem

Sellers often need liquidity upfront before auctions complete. Buyers sometimes hesitate to bid when they don't know if sellers will follow through. Existing NFT auctions don't integrate credit or loan markets.

Solution: Auction-Backed Lending

Our protocol allows sellers to borrow stablecoins upfront, collateralized by their domain NFTs locked in auction contracts. Lenders fund up to 30-50% of estimated floor prices with automatic settlement from auction proceeds.

Workflow

1. **List & Collateralize**: Seller lists domain for Dutch auction, NFT locked in smart contract, optionally requests advance loan
2. **Funding**: Lenders/protocol provide stablecoin liquidity with encoded terms (APR, repayment deadline, liquidation trigger)
3. **Auction Runs**: Dutch auction proceeds normally, buyers bid with confidence (NFT collateralized)
4. **Settlement**:
    - **Success**: Loan + interest auto-repaid from proceeds, remainder to seller
    - **Failure**: Loan defaults, NFT ownership transfers to lenders

Benefits

- **Sellers**: Get immediate liquidity instead of waiting for auction results, confidence to list high-value domains without cashflow pressure
- **Lenders**: Gain yield if auction succeeds, gain NFT at discounted price if auction fails
- **Buyers**: More secure - auction integrity guaranteed by collateralized domain
- **Protocol**: Introduces new credit layer around domain auctions, higher liquidity → higher auction participation

Benefits

- **Sellers**: Immediate liquidity, confidence to list high-value domains without cashflow pressure
- **Lenders**: Yield from successful auctions, discounted NFT acquisition on failures

- **Buyers**: Enhanced security through collateralized domains
- **Protocol**: New credit layer increases liquidity and participation

Example Scenario

- Seller lists premium.eth with 10,000 USDC reserve
- Requests 4,000 USDC loan (40% of reserve)
- Lenders fund loan, NFT locked
- Auction clears at 12,000 USDC
- Loan repaid: 4,000 + 200 interest = 4,200 USDC
- Seller receives: 7,800 USDC
- **Failure case**: Lenders acquire premium.eth NFT as repayment

System 4: Domain Voting Contest

Problem

Domain marketplaces lack community engagement and social discovery mechanisms. Users need ways to discover trending domains while earning rewards for participation.

Solution: Gamified Social Layer

Introduce a voting contest where community members stake tokens to vote for favorite domains, earning long-term staking rewards while boosting domain visibility through social discovery.

Key Features

1. Contest Lifecycle

- Only one contest active at any given time
- Created by hoster with configurable parameters:
  - startTime: Contest start timestamp
  - endTime: Contest end timestamp
  - minStake: Minimum token stake required (e.g., 100 tokens)
  - stakeMultiplier: Maximum stake multiplier (e.g., 5 → maxStake = minStake × 5)

2. Voting Mechanism

- Users can vote for up to 3 domains per contest
- Vote weight equals staking amount (not time-cumulated)
- Staking bounds: minStake ≤ stake ≤ minStake × multiplier
- Votes locked until contest ends
- Each wallet can participate only once per contest

3. Anti-Sybil & Fairness Measures

- Token staking required to prevent spam voting
- Staking caps prevent whale dominance
- One participation per wallet ensures fair distribution
- On-chain verification prevents manipulation

4. Voter Rewards System

- **Staking Points Formula**: StakingPoints = StakeAmount × TimeStaked
- Encourages early participation and long-term engagement
- **Reward Types**:
    - NFT badges for early voters
    - Fee discounts in future auctions/sales
    - Priority whitelists for next contests
    - Protocol governance rights

5. Contest Ranking & Transparency

- Domain score = sum of all vote stakes
- Real-time ranking updates
- Public and verifiable on-chain results
- Transparent reward distribution

Example Flow

1. **Contest Creation**: Hoster launches 10-day contest (minStake=100, multiplier=5)
2. **Voting Phase**:
    - Alice stakes 300 tokens → votes for domains A, B, C
    - Bob stakes 200 tokens → votes for domains A, B
    - Carol stakes 500 tokens → votes for domain C
3. **Contest Resolution**:
    - Domain A: 500 votes, Domain B: 500 votes, Domain C: 800 votes
    - Staking points: Alice=3,000, Bob=2,000, Carol=5,000
4. **Reward Distribution**: Users redeem points in next auction round

Benefits

- **Domain Owners**: Increased visibility, ranking-based promotion, higher sale probability
- **Voters**: Tangible rewards (NFTs, discounts, governance rights)

- **Marketplace**: Enhanced engagement, token utility, fair ecosystem governance
- **Community**: Social discovery, gamified participation, network effects

Technical Implementation

- **Staking Integration**: Leverages existing token economics
- **Time-Weighted Rewards**: Encourages sustained participation
- **On-Chain Transparency**: Verifiable results and fair distribution
- **Scalable Architecture**: Supports multiple concurrent contests

Benefits

- **Domain Owners**: Increased visibility, ranking-based promotion, higher chance of sale
- **Voters**: Tangible rewards (NFTs, discounts, points) for participation
- **Marketplace**: Enhanced engagement, token utility, fair ecosystem governance
- **Community**: Social discovery, gamified participation, network effects

Gamified Social Layer

- Community voting for favorite domains through token staking
- Earn staking rewards while boosting domain visibility
- Anti-Sybil measures ensure fair participation

Contest Features

- **Single Active Contest**: Only one contest runs at a time
- **Staking-Based Voting**: Vote weight equals staked amount
- **Bounded Stakes**: $\text{minStake} \leq \text{stake} \leq \text{minStake} \times \text{multiplier}$
- **Multiple Votes**: Up to 3 domains per participant
- **Time-Cumulated Rewards**: $\text{StakingPoints} = \text{StakeAmount} \times \text{TimeStaked}$

Voting Mechanism

- **Domain Ranking**: Score = sum of all vote stakes
- **Locked Votes**: Stakes locked until contest ends
- **Fair Distribution**: Rewards for fee discounts and priority access

Contract Architecture

Core Contracts

System 1 - Hybrid Batch Auctions:

- HybridDutchAuction.sol - Batch auction logic with gamification
- LoyaltyNFT.sol - Gamification rewards and loyalty points

System 2 - Premium Domain + Betting:

- DomainAuctionBetting.sol - Independent single-domain auctions with 4-tier betting

System 3 - Auction-Backed Lending:

- AuctionBackedLending.sol - Lending protocol integrated with batch auctions

System 4 - Domain Voting Contest:

- VotingContest.sol - Contest management, voting, staking, and reward distribution

Shared:

- IOwnershipToken.sol - Interface for Doma domain NFTs

Key Functions

System 1: Hybrid Batch Auction Functions

```
function createBatchAuction(
    IOwnershipToken nftContract,
    uint256[] memory tokenIds,
    uint256 startPrice,
    uint256 reservePrice,
    uint256 priceDecrement,
    uint256 duration,
    uint256 rewardBudgetBps,
    uint256 royaltyIncrement,
```

```
    address paymentToken
) external returns (uint256)

function placeSoftBid(uint256 auctionId, uint256 threshold, uint256 desiredCount) external payable
function placeHardBid(uint256 auctionId, uint256 desiredCount) external payable
function processConversions(uint256 auctionId) external
```

System 2: Premium Single Domain + Betting Functions

```
// Create single domain auction with betting price thresholds
function createSingleDomainAuction(uint256 tokenId, uint256 startPrice, uint256 reservePrice, uint256 priceDecrement, uint256 duration, uint256 highPrice, uint256 lowPrice) external

// Place bid on single domain (ends auction immediately)
function placeBid(uint256 auctionId) external payable

// Create betting pool with 4 price categories
function createBettingPool(uint256 auctionId, uint256 commitDuration, uint256 revealDuration) external

// Commit bet with hash of (choice, amount, secret)
function commitBet(uint256 auctionId, bytes32 commitHash, uint256 amount) external

// Reveal committed bet (choice: 3=Above High, 2=High~Low, 1=Below Low, 0=Uncleared)
function revealBet(uint256 auctionId, uint8 choice, uint256 amount, uint256 secret) external

// Settle betting after auction ends
function settleBetting(uint256 auctionId) external

// Owner functions
function setCuts(uint256 _sellerCut, uint256 _buyerCut, uint256 _protocolCut, uint256 _winnerCut) external onlyOwner
```

System 3: Auction-Backed Lending Functions

```
function requestLoan(
    uint256 auctionId,
    uint256 amount,
    uint256 interestBps,
    uint256 durationDays
) external returns (uint256)


function fundLoan(uint256 loanId, uint256 amount) external
function repayLoan(uint256 loanId) external
function checkAndLiquidate(uint256 loanId) external
```

System 4: Domain Voting Contest Functions

```
function listDomain(uint256 domainId) external
function createContest(uint256 startTime, uint256 endTime, uint256 minStake, uint256 multiplier) external
function vote(uint256[] calldata domainIds, uint256 stakeAmount) external
function endContest() external
function unlistDomain(uint256 contestId, uint256 domainId) external
function getStakingPoints(uint256 contestId, address user) external view returns (uint256)
function unstake(uint256 contestId) external
function getDomainVotes(uint256 contestId, uint256 domainId) external view returns (uint256)
function getAllDomainVotes(uint256 contestId) external view returns (uint256[] memory, uint256[] memory)
```

Examples

Example 1: Hybrid Batch Portfolio Auction with Gamification

Setup:

- Item: 100-domain bundle
- Start price: 1,000 USDC (Dutch, linearly down)
- Reserve floor: 700 USDC
- Reward budget: 1% of final sale, only if cleared
- Bond: 0.2% of intended spend

Early Phase:

- Alice: soft bid for 10% of bundle, threshold = 900 → bond posted
- Bob: soft bid 5%, threshold = 860 → bond posted
- Carol: soft bid 40%, threshold = 820 → bond posted
- Dana: soft bid 50%, threshold = 780 → bond posted

Price Progression:

- At 900: Alice auto-converts (10%). Cumulative = 10% — continue
- At 860: Bob auto-converts (5%). Cumulative = 15% — continue
- At 820: Carol auto-converts (40%). Cumulative = 55% — continue
- At 780: Dana auto-converts (50%). Cumulative = 105% ≥ 100% → auction clears at 780

Settlement:

- Pro-rata fill at clearing price (if over-subscribed)
- Bonds returned
- Rewards minted (since sale cleared):
    - Alice (earliest, highest price distance) gets largest share of points
    - Dana gets less (later threshold), even though she cleared the auction

```
// Create batch auction
createBatchAuction(
  ownershipToken,
  [1,2,3,...,100],  // 100 domain token IDs
  1000e18,        // 1000 USDC start price
  700e18,         // 700 USDC reserve
  1e18,           // 1 USDC per block decrement
  300,            // 300 blocks duration
  100,            // 1% reward budget (100 bps)
  0,              // No reverse royalty
  address(0)      // ETH payments
);
```

```
// Alice places early soft bid
placeSoftBid{value: 1.8e18}(auctionId, 900e18, 10); // 10 tokens at 900, bond = 1.8 USDC
```

Example 2: Premium Domain Auction with 4-Tier Betting

Setup:

- Single premium domain with price range betting
- Bettors wager on final price category
- 4 betting tiers: Above High, High~Low Range, Below Low, Uncleared

```
// Create single domain auction with betting thresholds
createSingleDomainAuction(tokenId, 100e18, 50e18, 0.5e18, 3600, 80e18, 60e18);
// highPrice = 80 ETH, lowPrice = 60 ETH

// Create betting pool
createBettingPool(auctionId, 3600, 1800); // 1hr commit, 30min reveal

// Commit bets (hidden)
bytes32 hash1 = keccak256(abi.encodePacked(uint8(3), 100e18, 12345)); // bet >80 ETH
bytes32 hash2 = keccak256(abi.encodePacked(uint8(2), 50e18, 67890)); // bet 60-80 ETH
commitBet(auctionId, hash1, 100e18);
commitBet(auctionId, hash2, 50e18);

// Someone bids on auction
placeBid{value: 75e18}(auctionId); // Auction clears at 75 ETH (category 2)

// Reveal after auction ends
revealBet(auctionId, 3, 100e18, 12345); // Wrong prediction
revealBet(auctionId, 2, 50e18, 67890); // Correct prediction (60-80 ETH range)

// Settle betting
```

```
settleBetting(auctionId); // Category 2 bettors win 90% of pool
```

Betting Categories:

- **Category 3**: Final price > High Price (above 80 ETH)
- **Category 2**: Low Price ≤ Final price ≤ High Price (60-80 ETH)
- **Category 1**: Final price < Low Price (below 60 ETH)
- **Category 0**: Auction fails to clear (no sale)

Example 3: Auction-Backed Lending

Setup:

- Seller lists premium domain with reserve price 10,000 USDC
- Seller requests 4,000 USDC loan (40% of reserve) at 5% APR for 7 days

// Request loan against auction

```
uint256 loanId = requestLoan(auctionId, 4000e18, 500, 7); // 4000 USDC, 5% APR, 7 days


// Lenders fund the loan
fundLoan(loanId, 2000e18); // Lender A funds 2000 USDC
fundLoan(loanId, 2000e18); // Lender B funds 2000 USDC


// Loan disbursed to seller automatically when fully funded


// If auction clears at 12,000 USDC
repayLoan(loanId); // Seller repays 4,200 USDC (principal + 5% interest)


// If auction fails
checkAndLiquidate(loanId); // NFT transferred to lenders
```

Example 4: Domain Voting Contest

Setup:

- Contest: 10 days, minStake = 100 tokens, multiplier = 5
- Domains: A (ID 0), B (ID 1), C (ID 2) listed for voting

// Domain owners list their domains

```
listDomain(0); // Owner of domain A lists it
listDomain(1); // Owner of domain B lists it
listDomain(2); // Owner of domain C lists it

// Create contest
createContest(block.timestamp, block.timestamp + 10 days, 100e18, 5);

// Users vote (staking happens automatically)
vote([0, 1, 2], 300e18); // Alice votes for A, B, C with 300 stake
vote([0, 1], 200e18); // Bob votes for A, B with 200 stake
vote([2], 500e18); // Carol votes for C with 500 stake

// After 10 days, end contest
endContest();

// Rankings: A=500, B=500, C=800
// Staking points: Alice=300*10=3000, Bob=200*10=2000, Carol=500*10=5000
// Domains returned to owners
```

// Users can unstake after contest

unstake(); // Return staked tokens

Example 4: Domain Voting Contest

Setup:

- Contest: 10 days duration, minStake = 100 tokens, multiplier = 5
- Domains: A (ID 0), B (ID 1), C (ID 2) listed for voting
- Reward pool: 10,000 tokens for distribution

// Domain owners list their domains

```
listDomain(0); // Domain A owner lists domain
```

```
listDomain(1); // Domain B owner lists domain
listDomain(2); // Domain C owner lists domain

// Create contest
createContest(
   block.timestamp,                 // startTime
   block.timestamp + 10 days,       // endTime
   100e18,                          // minStake (100 tokens)
   5                                // multiplier (max stake = 500 tokens)
);

// Users participate in voting
vote([0, 1, 2], 300e18); // Alice votes for A, B, C with 300 stake
vote([0, 1], 200e18);    // Bob votes for A, B with 200 stake
vote([2], 500e18);       // Carol votes for C with 500 stake

// Check rankings during contest
(uint256[] memory domainIds, uint256[] memory votes) = getAllDomainVotes(contestId);
// Returns: domainIds=[0,1,2], votes=[500,500,800]

// After contest ends
endContest(); // Final rankings: A=500, B=500, C=800

// Check staking points
uint256 alicePoints = getStakingPoints(contestId, aliceAddress); // 300 * 10 days = 3,000
uint256 bobPoints = getStakingPoints(contestId, bobAddress);     // 200 * 10 days = 2,000
uint256 carolPoints = getStakingPoints(contestId, carolAddress); // 500 * 10 days = 5,000

// Users unstake and claim rewards
unstake(contestId); // Return staked tokens and claim rewards
```

Contest Results:

- **Domain Rankings**: C (800 votes) > A (500 votes) = B (500 votes)
- **Staking Rewards**: Carol earns highest points, gets priority access to next contest
- **Community Benefits**: Increased domain visibility and marketplace engagement

Deployment

Prerequisites

git clone https://github.com/0xawang/DomaAuction.git

cd DomaAuction

npm install

Compile

npx hardhat compile

Deploy to Doma Testnet

 # Set PRIVATE_KEY in .env

cp .env.example .env

Deploy scripts

"deploy": "hardhat run scripts/deploy.js --network doma",

"deploy:betting": "hardhat run scripts/deploy-betting.js --network doma",

"deploy:lending": "hardhat run scripts/deploy-lending.js --network doma",

"deploy:voting-contest": "hardhat run scripts/deploy-voting-contest.js --network doma"

Contract Addresses

- **Doma OwnershipToken**: 0x424bDf2E8a6F52Bd2c1C81D9437b0DC0309DF90f

System 1 - Hybrid Batch Auctions:

- **HybridDutchAuction**: 0xE680A0F580f742a536B33C142757b4C8BE5CfB40
- **LoyaltyNFT**: 0x04B36cADFD85F2561c0e8A676E0aCe5cBA8c7485

System 2 - Premium Single Domain + Betting:

- **DomainAuctionBetting**: 0x5A11663fc4cBfa62E01C3bbCfDb10f37549B38D2

System 3 – Auction Backed Lending:

- **AuctionBackedLending**: 0x11814d7d0927EEAcE1801356254F8Ea258da2385

System 4 – Domain Voting Contest:

- **VotingContest**: 0xE507304592D06F67b86DE449f29A2029010F2d7B

Events

System 1: Batch Auction Events

```
event AuctionCreated(uint256 indexed auctionId, address seller, uint256 startPrice, uint256 reservePrice, bool hasReverseRoyalty);

event SoftBidPlaced(uint256 indexed auctionId, address bidder, uint256 threshold, uint256 count, uint256 bond);

event SoftBidConverted(uint256 indexed auctionId, address bidder, uint256 price, uint256 count);

event AuctionCleared(uint256 indexed auctionId, uint256 clearingPrice, uint256 totalRewards, uint256 royaltyAmount);
```

System 2: Premium Auction + Betting Events

```
event AuctionCreated(uint256 indexed auctionId, address seller, uint256 tokenId, uint256 startPrice);

event BidPlaced(uint256 indexed auctionId, address bidder, uint256 price);

event AuctionEnded(uint256 indexed auctionId, bool cleared, address winner, uint256 finalPrice);

event BettingPoolCreated(uint256 indexed auctionId, uint256 commitDeadline, uint256 revealDeadline);

event BetCommitted(uint256 indexed auctionId, address indexed bettor, bytes32 commitHash, uint256 amount);

event BetRevealed(uint256 indexed auctionId, address indexed bettor, uint8 choice, uint256 amount);

event BettingSettled(uint256 indexed auctionId, uint8 auctionResult, uint256 totalPool);
```

System 3: Auction-Backed Lending Events

```
event LoanRequested(uint256 indexed loanId, uint256 indexed auctionId, address borrower, uint256 amount, uint256 interestBps);

event LoanFunded(uint256 indexed loanId, address lender, uint256 amount);

event LoanRepaid(uint256 indexed loanId, uint256 totalRepayment);

event LoanLiquidated(uint256 indexed loanId, address liquidator);
```

System 4: Domain Voting Contest Events

```
event ContestCreated(uint256 indexed contestId, uint256 startTime, uint256 endTime, uint256 minStake, uint256 multiplier);

event DomainListed(uint256 indexed domainId, address indexed owner);

event Voted(address indexed user, uint256[] domainIds, uint256 stakeAmount);

event ContestEnded(uint256 indexed contestId, uint256[] rankedDomains, uint256[] scores);

event Unstaked(address indexed user, uint256 amount);
```